SESSION 8

iOS DEVELOPMENT

THE MANY WAYS TO

PERSIST DATA

WHAT IS DATA PERSISTENCE?

- Data stored in memory will be released when the user quits your app.
    - *Example: an array of objects on a view controller*
- Most apps need to persist (save) data so that users can leave and come back without losing information.

OPTIONS FOR DATA PERSISTENCE

- Store data on the device:
  - User Defaults
  - File Manager

  } We'll cover these two!
- Core Data

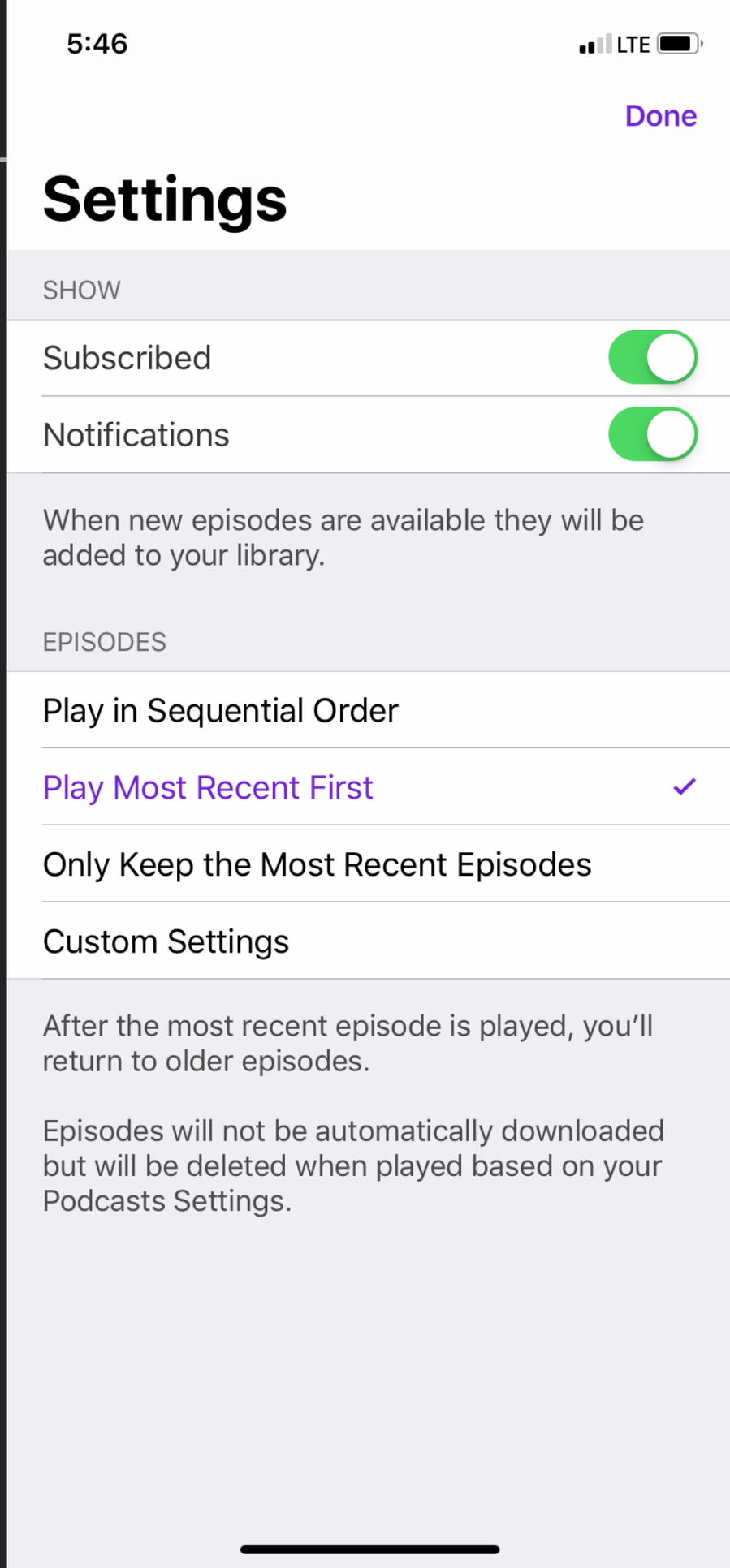
OPTIONS FOR DATA PERSISTENCE

- Store data remotely:
  - Your own server
  - Someone else's server

  - iCloud
  - Firebase

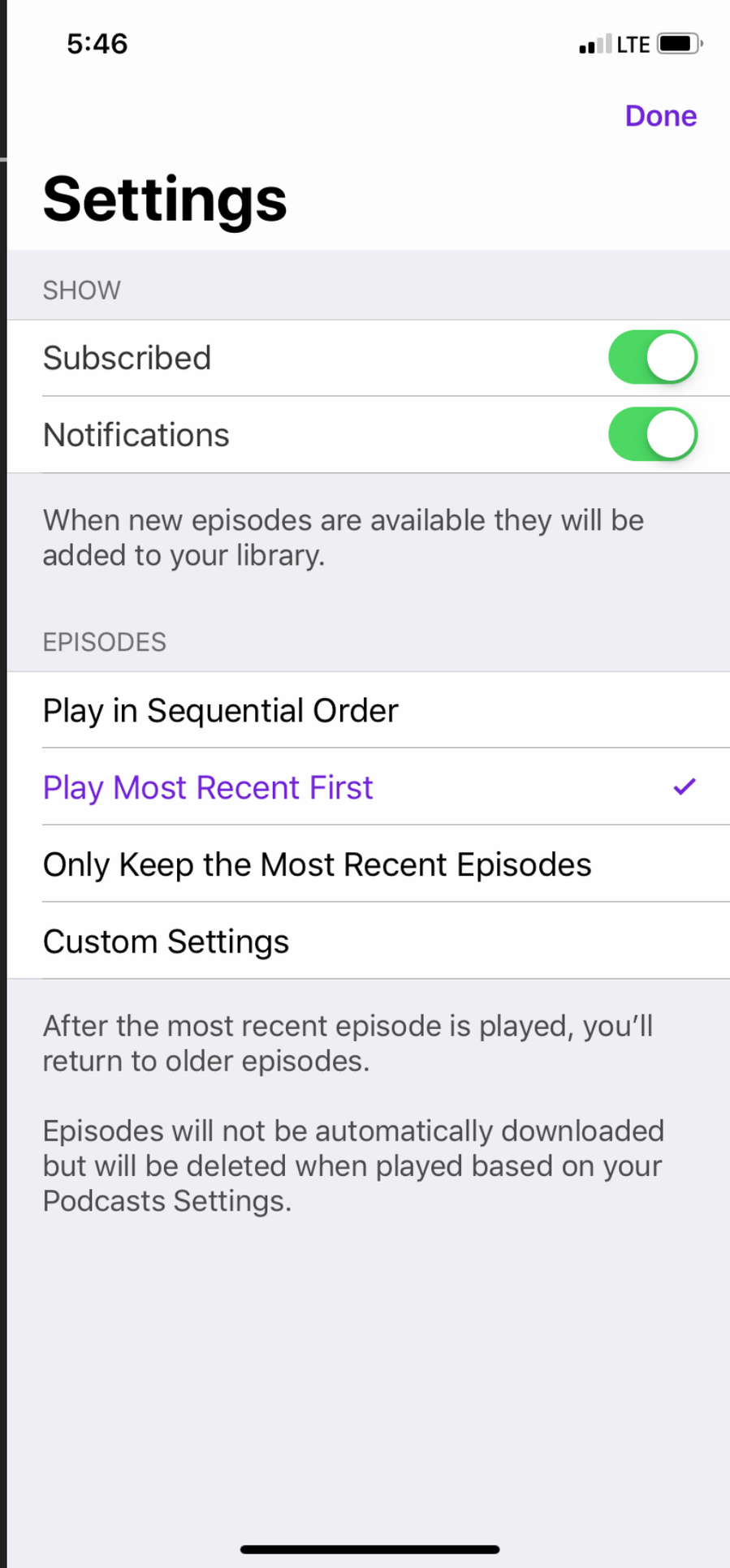
USER DEFAULTS

- Intended to store small amounts of data
  - Example: order of episodes



USER DEFAULTS

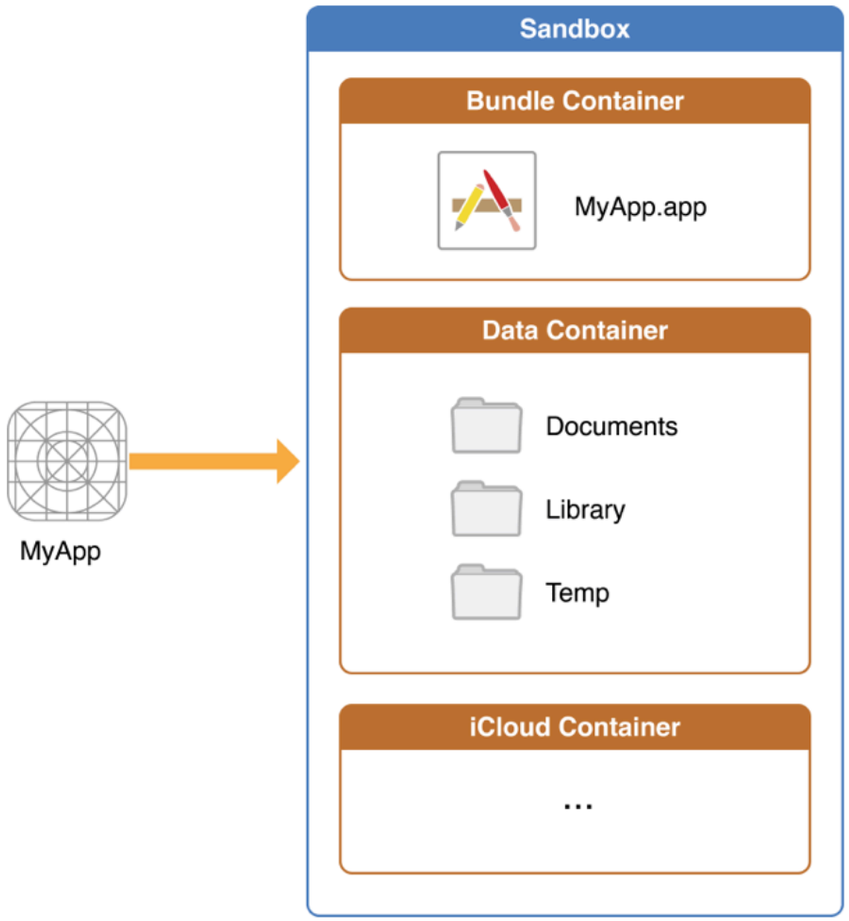
- Typically used to restore an app to the user's preferred default state
- But... you can store any values here



FILE MANAGER

- Every app can create and access files inside its sandbox directory
- An app cannot access another app's files

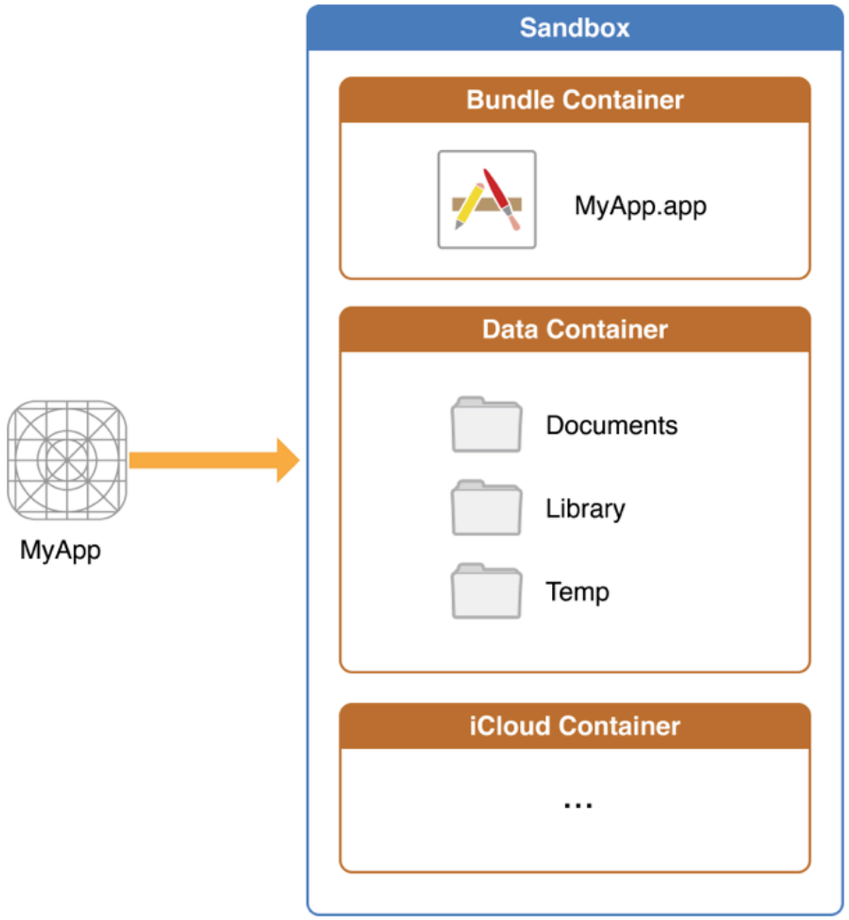
Figure 1–1 An iOS app operating within its own sandbox directory



FILE MANAGER

- Documents
  - Files containing user data
- Library
  - Files that don't contain user data
- Temp
  - Temporary files

Figure 1–1 An iOS app operating within its own sandbox directory

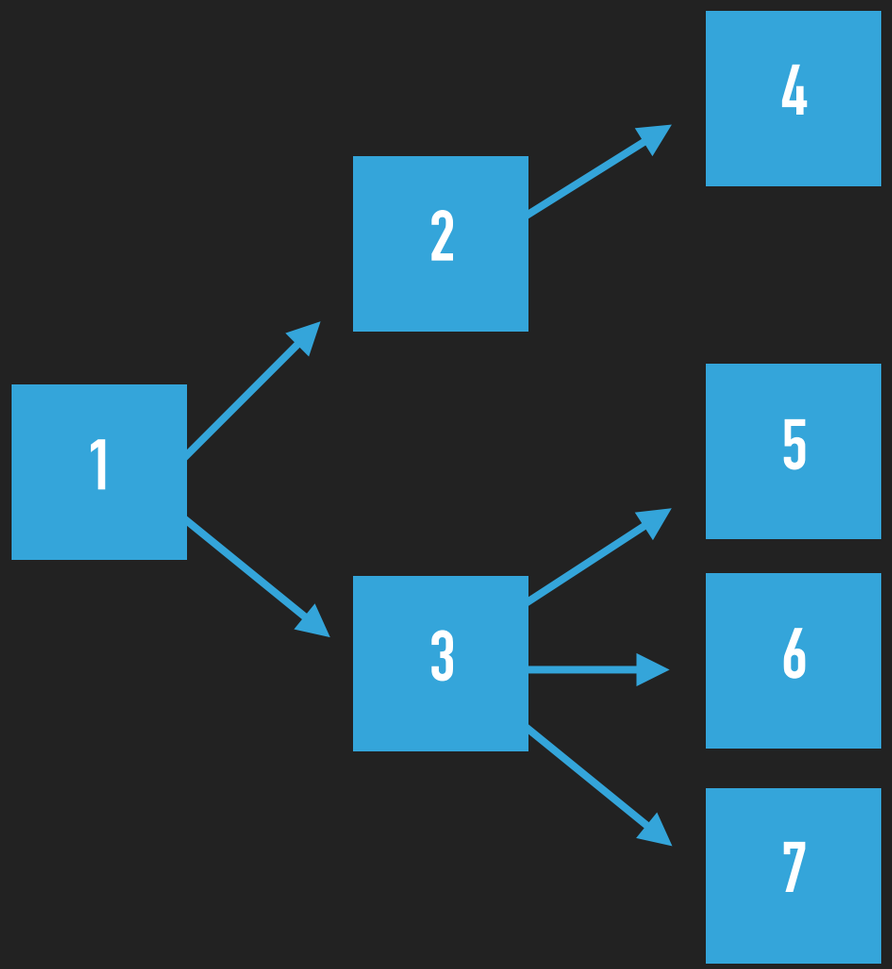


FILE MANAGER

- You can store any type of file
  - Images
  - Text
  - Audio
  - Archived data

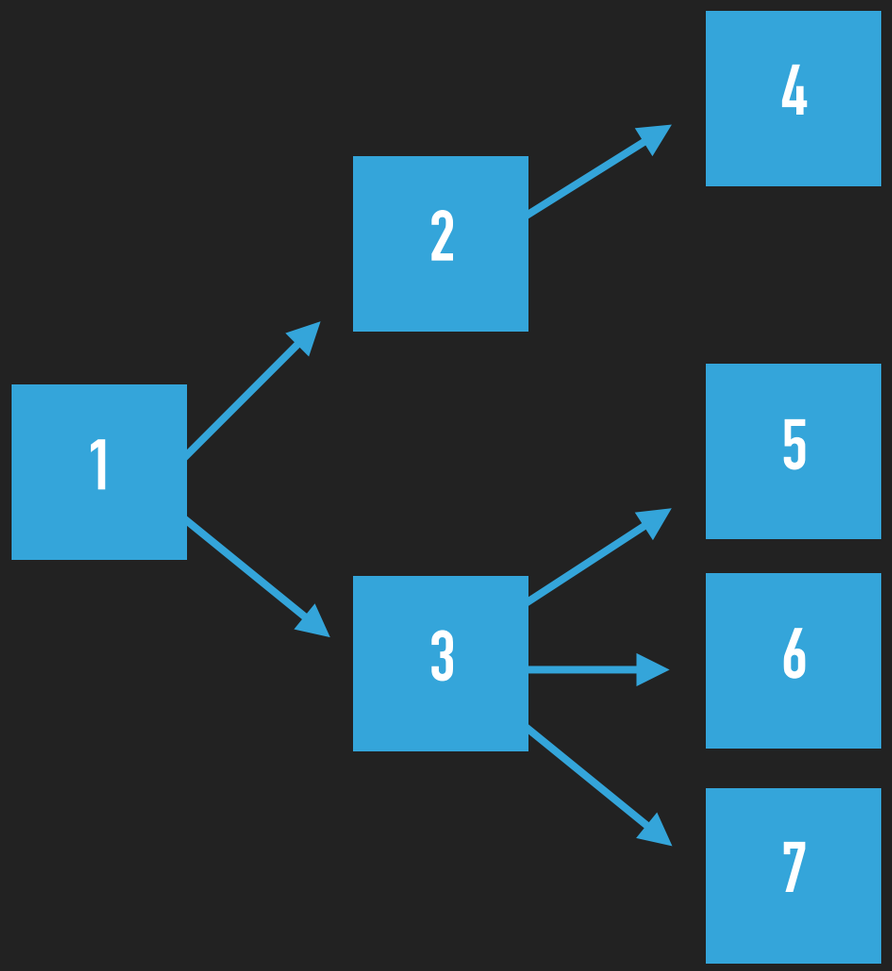
CORE DATA

- Useful if you have many objects related to each other in complex ways



CORE DATA

- Manages an object graph that you can traverse and query



CORE DATA

- Data persistence
    - Tracks which objects in memory have been created, deleted or changed
- Performs automatic data migrations

CORE DATA

When it comes to modeling, querying, traversing and persisting complex object graphs, there is no substitute for Core Data. Core Data is a big hammer, but not every problem is a nail—much less a sufficiently large nail.

~ NSHipster (https://nshipster.com/nscoding/)

iCLOUD

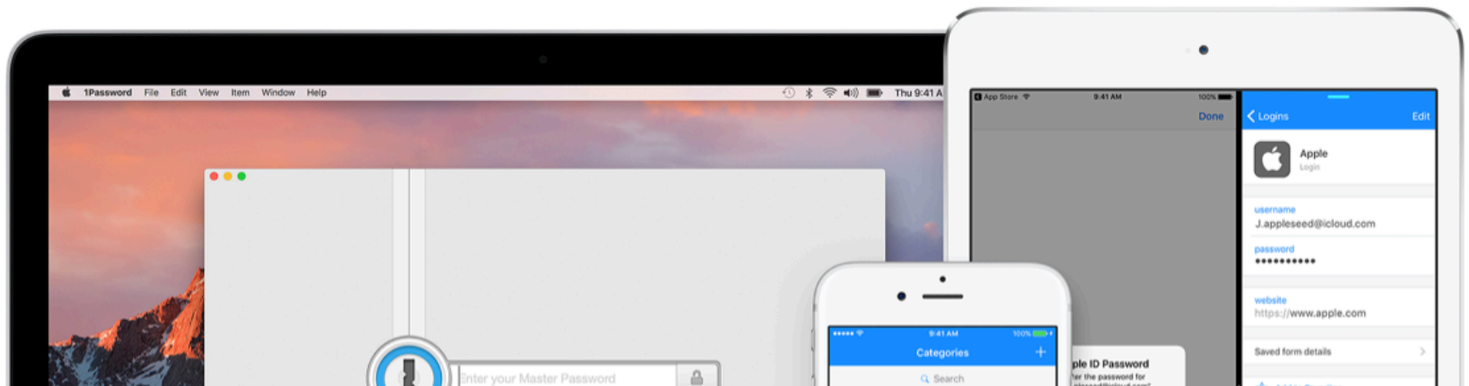
- Apple’s cloud-based persistence option

iCloud for Developers

Securely store your app's data and documents in iCloud — and keep them up to date across macOS, iOS, watchOS, tvOS, and the web. With iCloud, your users can access the information they want, wherever they want it.

CloudKit

CloudKit keeps your app data connected and up to date. And CloudKit JS makes it



iCLOUD

- Pros:
  - Reasonable amount of storage included with paid developer account
  - Automatically sync across a user's devices
  - Users don't have to create new accounts

iCLOUD

- Cons:
  - Data isn’t accessible outside of Apple’s ecosystem

MORE CONCEPTS IN

SWIFT

CONVENIENCE

INITIALIZERS

CONVENIENCE INITIALIZER

- A convenience initializer is one that calls another initializer on the same class
- This helps you avoid duplicate code
- Easier to read
- Less bug-prone

CONVENIENCE INITIALIZER

- An initializer that doesn’t call any other initializers is a designated initializer
- A class can have multiple designated and convenience initializers

CONVENIENCE INITIALIZER

```swift
class Person {

    let firstName: String
    let lastName: String
    let fullName: String

    // designated initializer
    init(firstName: String, lastName: String, fullName: String) {
        self.firstName = firstName
        self.lastName = lastName
        self.fullName = fullName
    }

    // convenience initializer
    convenience init(firstName: String, lastName: String) {
        let fullName = "\(firstName) \(lastName)"
        self.init(firstName: firstName, lastName: lastName, fullName: fullName)
    }

}
```

CONVENIENCE INITIALIZER MUST CALL THE DESIGNATED INITIALIZER

FAILABLE

INITIALIZERS

FAILABLE INITIALIZERS

- Sometimes you want to allow an initializer to fail to create an instance of an object
    - Example: `Int` has failable initializer that takes a string

```
38
39  let a = Int("400")
40  let b = Int("banana")
42
```

```
400
nil
```

FAILABLE INITIALIZERS

- Failable initializers always return an optional
    - If the initializer succeeds, the optional contains a value
    - Otherwise, the object is nil

```
38
39  let a = Int("400")                    400
40  let b = Int("banana")                 nil
42
```

FAILABLE INITIALIZERS

```
20  class Cat {
21
22      let name: String
23      let age: Int
24
25      // failable initializer
26      init?(name: String, age: Int) {
27          guard !name.isEmpty, age > 0 else { return nil }
28          self.name = name
29          self.age = age
30      }
31  }
32
```

INIT? INDICATES THAT THE INITIALIZER IS FAILABLE

ENUMS WITH

RAW VALUES

ENUMS WITH RAW VALUES

- Each case of an enum can be associated with a raw value

DECLARING THE RAW VALUE'S TYPE

```
3  enum State: String {
4      case illinois = "IL"
5      case indiana = "IN"
6      case michigan = "MI"
7      case minnesota = "MN"
8      case ohio = "OH"
9      case wisconsin = "WI"
10 }
11
```

ENUMS WITH RAW VALUES

- Raw values can be strings, characters or numbers
- Each case needs to have the same type of raw value (meaning, you can't mix and match `String` and `Int`)
- Each case needs a unique raw value (no duplicates)

ENUMS WITH RAW VALUES

- An enum with raw values automatically gets an initializer that receives a raw value as input
    - This initializer is failable!

```
14  let landOfLincoln = State(rawValue: "IL")      illinois
15  let sunshineState = State(rawValue: "FL")      nil
```

HANDLING ERRORS

DO-TRY-CATCH

REPRESENTING ERRORS

- Errors are types that conform to the `Error` protocol
- Enums work well for representing related error conditions

```
 8  enum CheckoutError: Error {
 9      case titleNotFound
10      case titleUnavailable
11  }
```

WHY THROW AN ERROR?

- Why throw an error when you could just return `nil`?

THROWING ERRORS

- A function that can throw an error instead of returning normally must be marked with `throws`

```
19      func checkout(title: String) throws -> Book {
20          var requestedBook: Book?
21          for book in books {
22              if book.title == title {
23                  requestedBook = book
24              }
25          }
26
27          guard let book = requestedBook else {
28              throw CheckoutError.titleNotFound
29          }
30
31          guard book.isAvailable else {
32              throw CheckoutError.titleUnavailable
33          }
34
35          return book
36      }
37
```

HANDLING ERRORS

- Use do-try-catch when calling a function that throws
  - do block must contain a function that throws
  - Mark throwing function with `try`
  - `catch` block only executes if an error is actually thrown

```
41  do {
42      let book1 = try library.checkout(title: "The DaVinci Code")
43      print(book1.title)
44  } catch (let error) {
45      print(error)
46  }
```

HANDLING ERRORS

- You can also choose to ignore potential errors
  - `try?` means return nil if an error is thrown
  - `try!` means crash if an error is thrown

```
48  let book2 = try? library.checkout(title: "Where the Wild Things Are")
49  let book3 = try! library.checkout(title: "The Hobbit")
50
```

SAVING BASIC TYPES WITH

USER DEFAULTS

SAVING A VALUE

- Associate a value with a key when saving

```
UserDefaults.standard.set(nameTextField.text, forKey: "name")
```

RETRIEVING A VALUE

- Retrieve the value using the same key

```
nameTextField.text = UserDefaults.standard.string(forKey: "name")
```

REMOVING A VALUE

- Remove the value for a key

```
UserDefaults.standard.removeObject(forKey: "name")
```

TYPES THAT CAN BE SAVED

- `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, `NSDictionary`
    - All of these types have toll-free bridging to their Swift counterparts

SETTING DEFAULT VALUES

Setting Default Values

```
func set(Any?, forKey: String)
```
Sets the value of the specified default key.

```
func set(Float, forKey: String)
```
Sets the value of the specified default key to the specified float value.

```
func set(Double, forKey: String)
```
Sets the value of the specified default key to the double value.

```
func set(Int, forKey: String)
```
Sets the value of the specified default key to the specified integer value.

```
func set(Bool, forKey: String)
```
Sets the value of the specified default key to the specified Boolean value.

```
func set(URL?, forKey: String)
```
Sets the value of the specified default key to the specified URL.

GETTING DEFAULT VALUES

Getting Default Values

```
func object(forKey: String) -> Any?
```
Returns the object associated with the specified key.

```
func url(forKey: String) -> URL?
```
Returns the URL associated with the specified key.

```
func array(forKey: String) -> [Any]?
```
Returns the array associated with the specified key.

```
func dictionary(forKey: String) -> [String : Any]?
```
Returns the dictionary object associated with the specified key.

```
func string(forKey: String) -> String?
```
Returns the string associated with the specified key.

```
func stringArray(forKey: String) -> [String]?
```
Returns the array of strings associated with the specified key.

```
func data(forKey: String) -> Data?
```
Returns the data object associated with the specified key.

```
func bool(forKey: String) -> Bool
```

REMOVING DEFAULT VALUES

Removing Defaults

```
func removeObject(forKey: String)
```

Removes the value of the specified default key.

SAVING CUSTOM TYPES WITH

USER DEFAULTS

SAVING CUSTOM TYPES

- What if I want to save an object that is not one of the supported types?
    - Supported types: `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, `NSDictionary`

SAVING CUSTOM TYPES

- To save an instance of a custom type, convert it into `NSData`
- Conforming to the `NSCoding` protocol makes it easy to convert between your custom type and `NSData`

NSCODING

- `NSCoding` has two methods that you must implement

```
protocol NSCoding {
    init?(coder aDecoder: NSCoder)
    func encode(with aCoder: NSCoder)
}
```

NSOBJECT

- You also need to subclass `NSObject` , which allows the object to be treated like an Objective C type

IMPLEMENTING NSCODING

```
class Book: NSObject, NSCoding {

    let title: String
    let author: String
    let category: BookCategory

    init?(title: String?, author: String?, category: BookCategory?) {•••}

    required convenience init?(coder aDecoder: NSCoder) {
        let title = aDecoder.decodeObject(forKey: "title") as? String
        let author = aDecoder.decodeObject(forKey: "author") as? String
        let categoryRawValue = aDecoder.decodeInteger(forKey: "category")
        let category = BookCategory(rawValue: categoryRawValue)

        self.init(title: title, author: author, category: category)
    }

    func encode(with aCoder: NSCoder) {
        aCoder.encode(title, forKey: "title")
        aCoder.encode(author, forKey: "author")
        aCoder.encode(category.rawValue, forKey: "category")
    }
}
```

NSKEYEDARCHIVER

- Use `NSKeyedArchiver` to convert the object into `Data`

```
do {
    let data = try NSKeyedArchiver.archivedData(withRootObject: books,
                                                requiringSecureCoding: false)

    UserDefaults.standard.set(data, forKey: "books")
} catch (let error) {
    print("Error saving books to user defaults: \(error)")
}
```

NSKEYEDUNARCHIVER

- Use `NSKeyedUnarchiver` to unarchive the `Data` back into your custom object

```
guard let data = UserDefaults.standard.data(forKey: "books") else { return [] }

var books: [Book]?
do {
    books = try NSKeyedUnarchiver.unarchiveTopLevelObjectWithData(data) as? [Book]
} catch (let error) {
    print("Error fetching books from user defaults: \(error)")
}

return books ?? []
```

SAVING TO DISK WITH

FILE MANAGER

FILE MANAGER

- “A convenient interface to the contents of the file system, and the primary means of interacting with it.”

  ~ Apple Documentation

FILE MANAGER

- Makes it easy to get the path to your app's Documents Directory, where you can store data to files

```
guard let documentDirectory = FileManager.default.urls(for: .documentDirectory,
                                                       in: .userDomainMask).first else { return }

let url = documentDirectory.appendingPathComponent("books")
```

FILE MANAGER

- Convert your custom objects to data with `NSKeyedArchiver`
- Write data to your new url in the Documents Directory

```
do {
    let data = try NSKeyedArchiver.archivedData(withRootObject: books, requiringSecureCoding: true)
    try data.write(to: url)
} catch (let error) {
    print("Error saving books to file: \(error)")
}
```

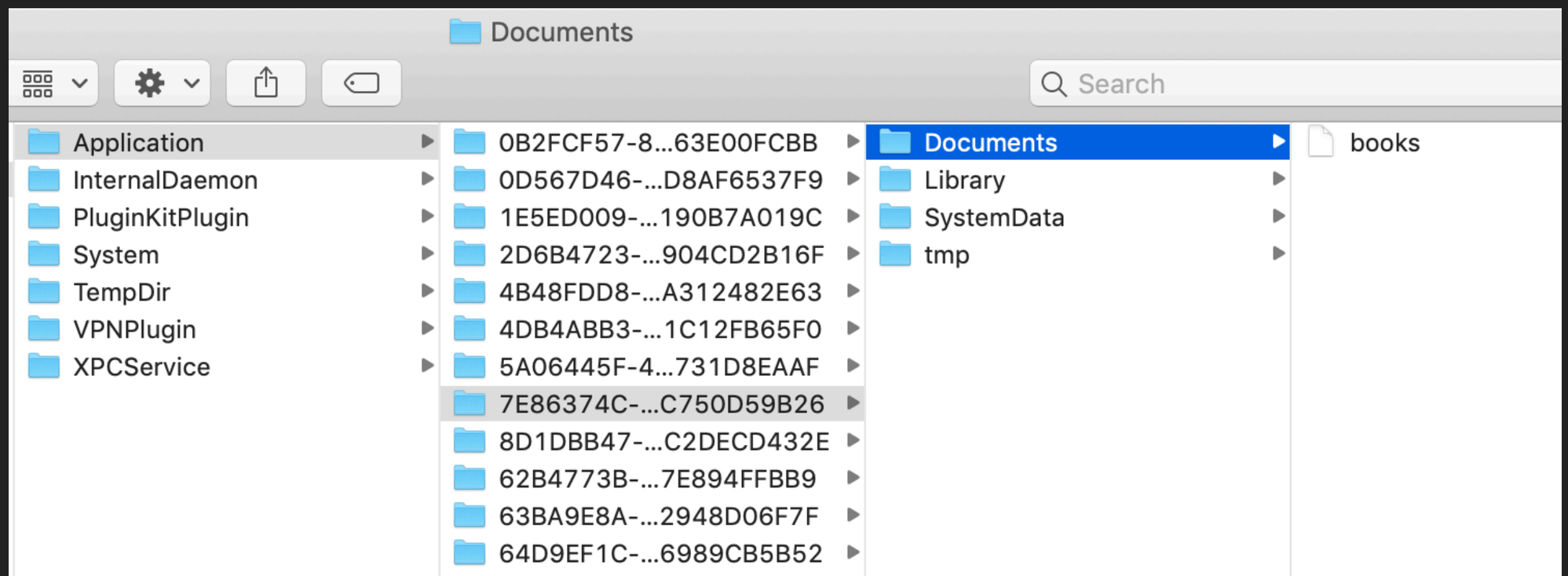
FILE MANAGER

- Fetch data from the file
- Convert the data back into your custom objects with `NSKeyedUnarchiver`

```
var books: [Book]?
do {
    let data = try Data(contentsOf: url)
    books = try NSKeyedUnarchiver.unarchiveTopLevelObjectWithData(data) as? [Book]
} catch (let error) {
    print("Error fetching books from file: \(error)")
}
```

TIP #1

- To navigate to your simulator's Document Directory:
  - Print the path to the Xcode console and copy it
  - In Finder, press Shift+Command+G and paste the path



TIP #2

- Deleting the app will delete its data from the Documents Directory and User Defaults
    - You'll need to do this if you change the structure of the data you're saving
- You can also manually delete a file from the Documents Directory if you're using a simulator

TIP #3

- Every time you run your app on a different simulator, you get a new sandbox
    - New Documents Directory
    - New User Defaults
- In other words, data is not shared between simulators

TIP #4

- Make sure to test that your app works the **first time** the user runs it
    - There won't be anything in the Documents Directory or User Defaults

BUILDING FORMS FOR

USER INPUT

CONTROLS FOR USER INPUT

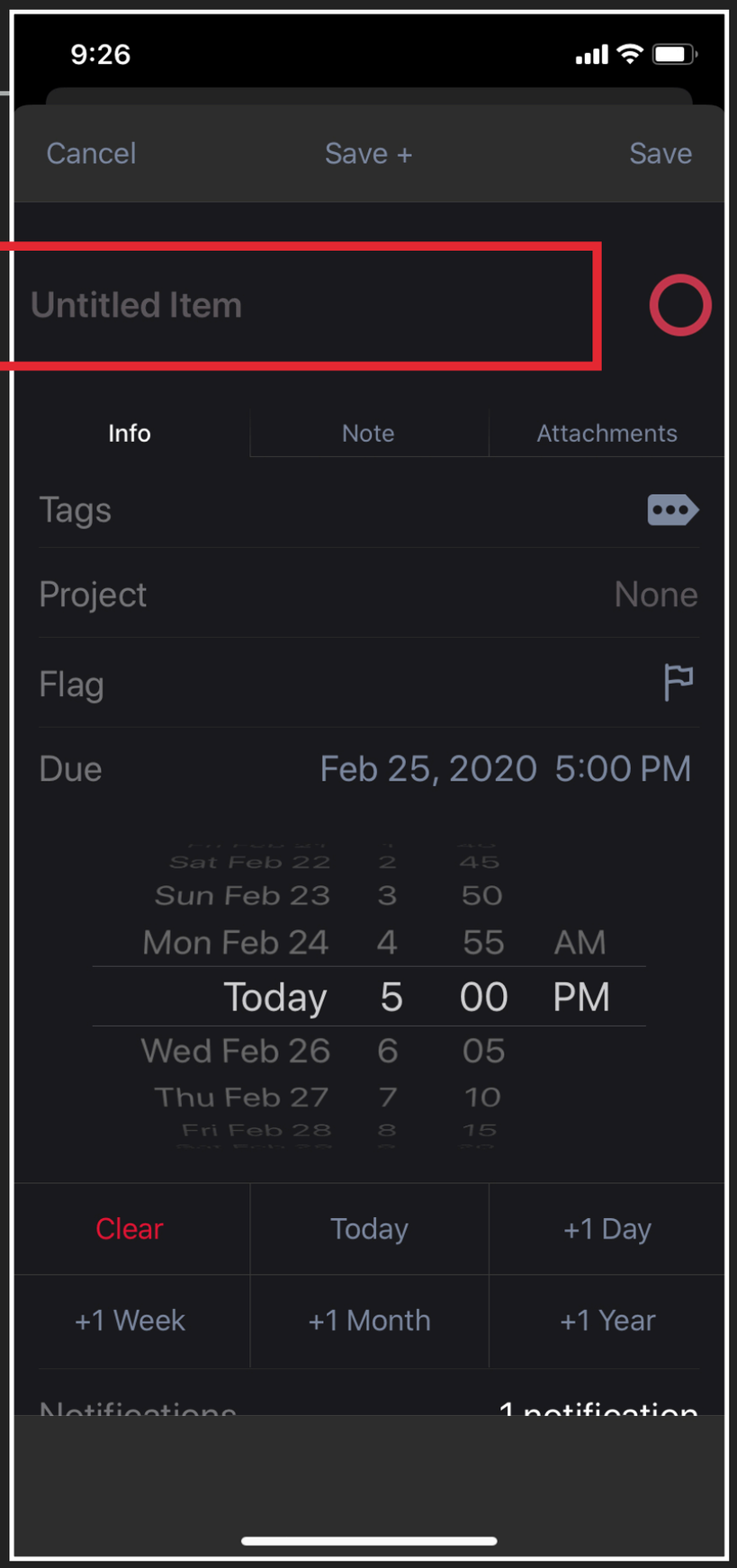
- iOS provides many controls that you can use to allow users to input data
- Examples:
  - Segmented control
  - Slider
  - Switch
  - Stepper
  - Date Picker

CONTROLS FOR USER INPUT

- There are three main controls that involve text input:
  - UITextField
  - UITextView
  - UISearchBar

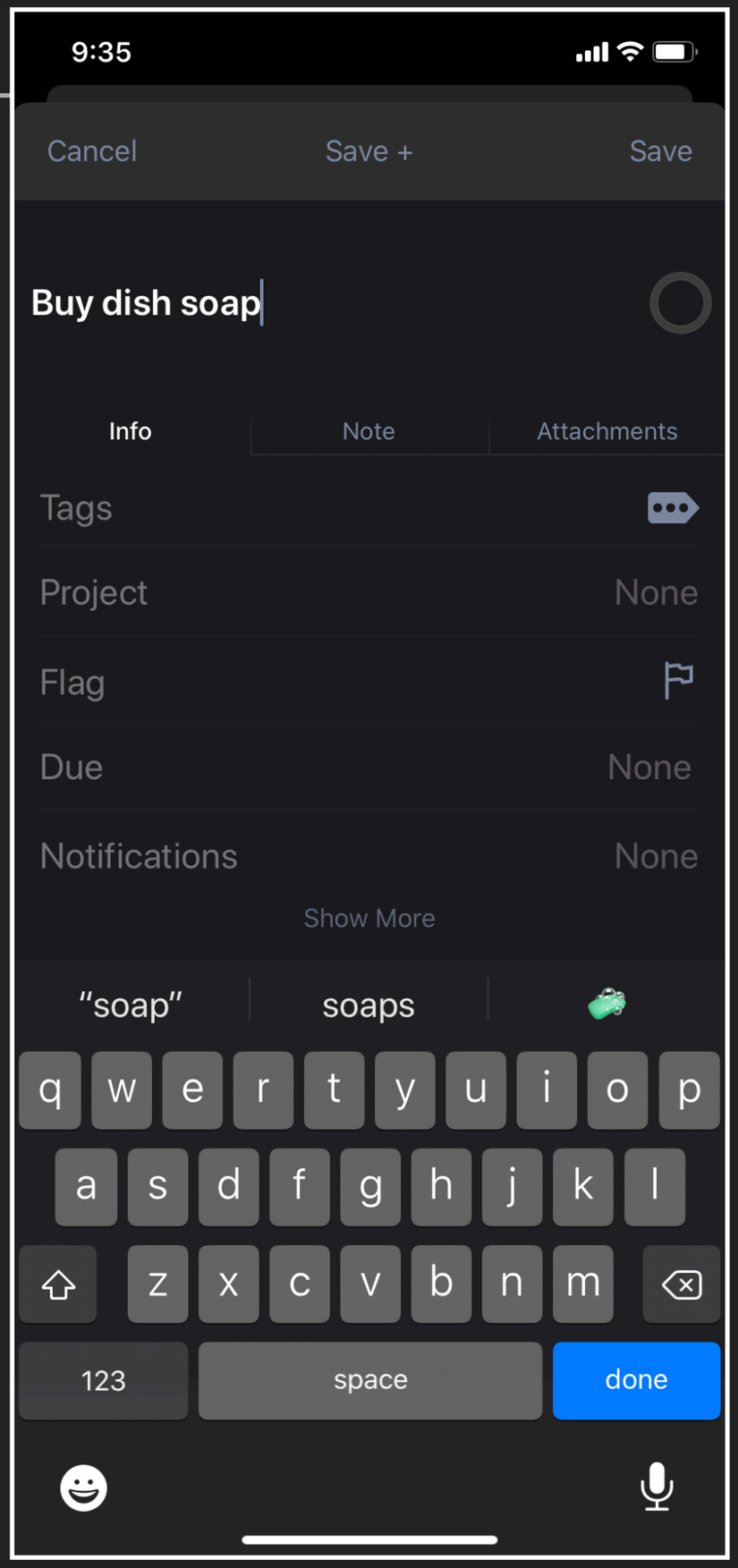
TEXT FIELD

- UITextField displays a single line of editable text



TEXT FIELD

- When the user taps inside of a text field, the keyboard automatically appears
- The text field becomes the first responder



OmniFocus

TEXT FIELD

- You can programmatically show and hide the keyboard by calling these methods:

```
textField.becomeFirstResponder()
textField.resignFirstResponder()
```

TEXT FIELD

- Even though the keyboard *shows* automatically, it does not *hide* automatically
- You may want to programmatically hide the keyboard when:
  - The user taps “Return”
  - The user taps away from the text field
  - The user drags down on the keyboard

TEXT FIELD

- Use UITextFieldDelegate to be notified of important events related to the text field
- For example:
  - Editing began or ended
  - Characters were added or deleted
  - User tapped the “Return” or “Clear” buttons

TEXT FIELD

- Use `textFieldShouldReturn` to dismiss the keyboard when the user taps on the “Return” button

```
32
33  extension AddBookViewController: UITextFieldDelegate {
34
35      func textFieldShouldReturn(_ textField: UITextField) -> Bool {
36          textField.resignFirstResponder()
37      }
38  }
39
```

TEXT FIELD

- Use a tap gesture recognizer to hide the keyboard when the user taps away from the text field
- Call `endEditing` on the view to have any text field in its view hierarchy resign first responder

```swift
@objc func handleTap(_ sender: UITapGestureRecognizer) {
    // Dismiss the keyboard when the user taps away from a text field
    tableView.endEditing(true)
}
```

TEXT FIELD

- If the text field is inside of a scroll view, you can set the scroll view's `keyboardDismissMode`

`case none`

The keyboard does not get dismissed with a drag.

`case onDrag`

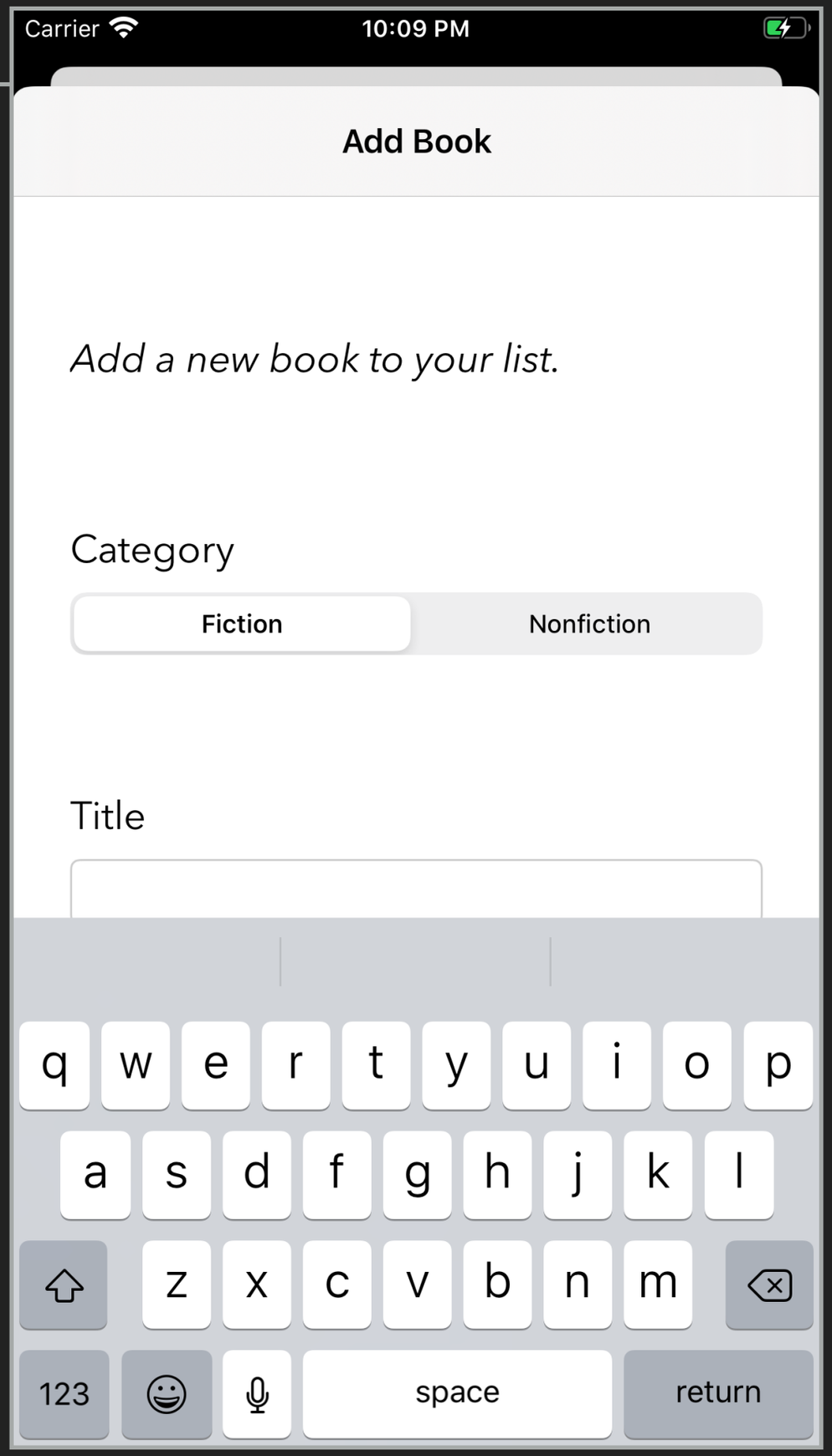
The keyboard is dismissed when a drag begins.

`case interactive`

The keyboard follows the dragging touch offscreen, and can be pulled upward again to cancel the dismiss.

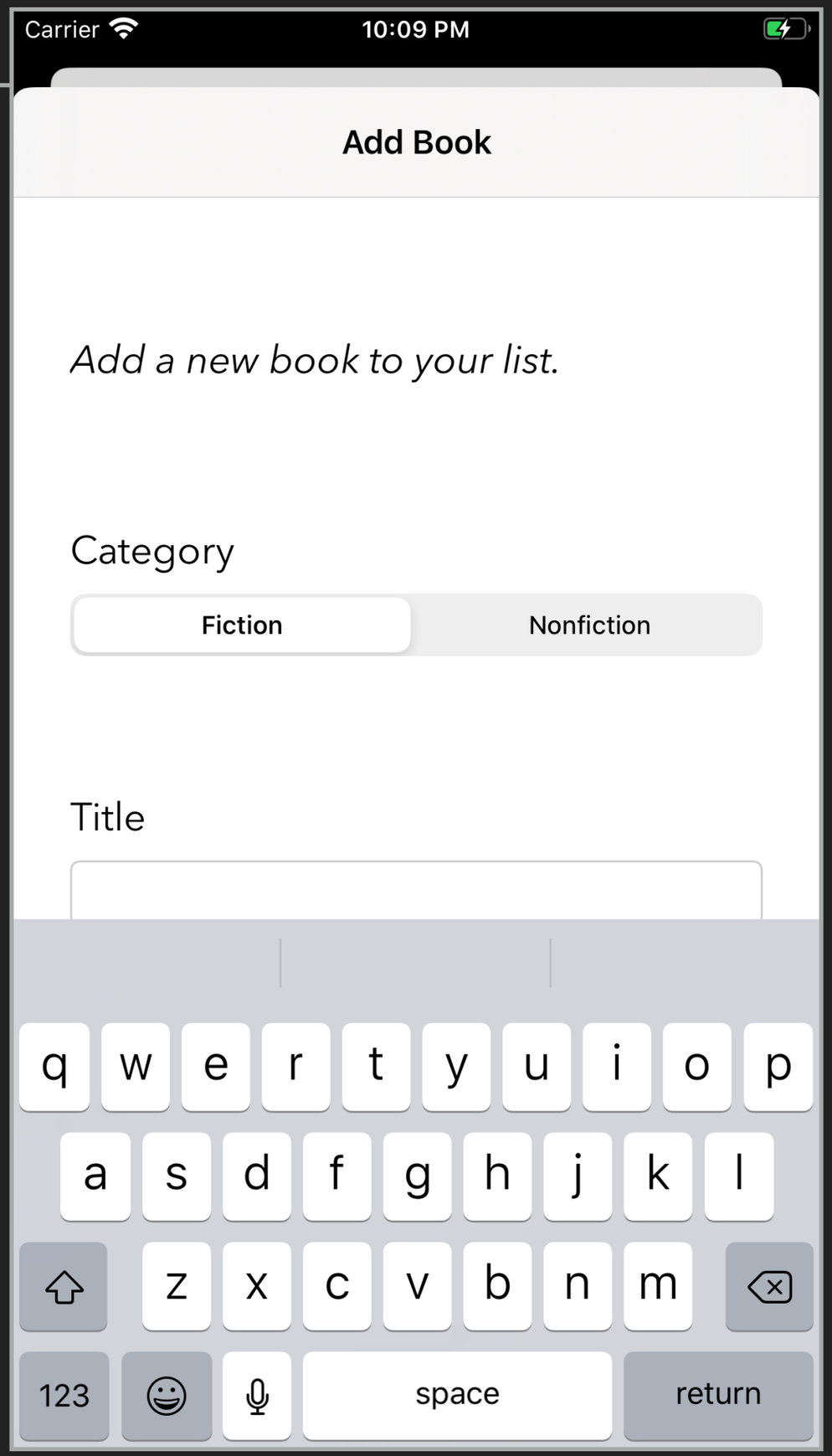
TEXT FIELD

- What do we do if the keyboard covers important parts of the user interface? 😢



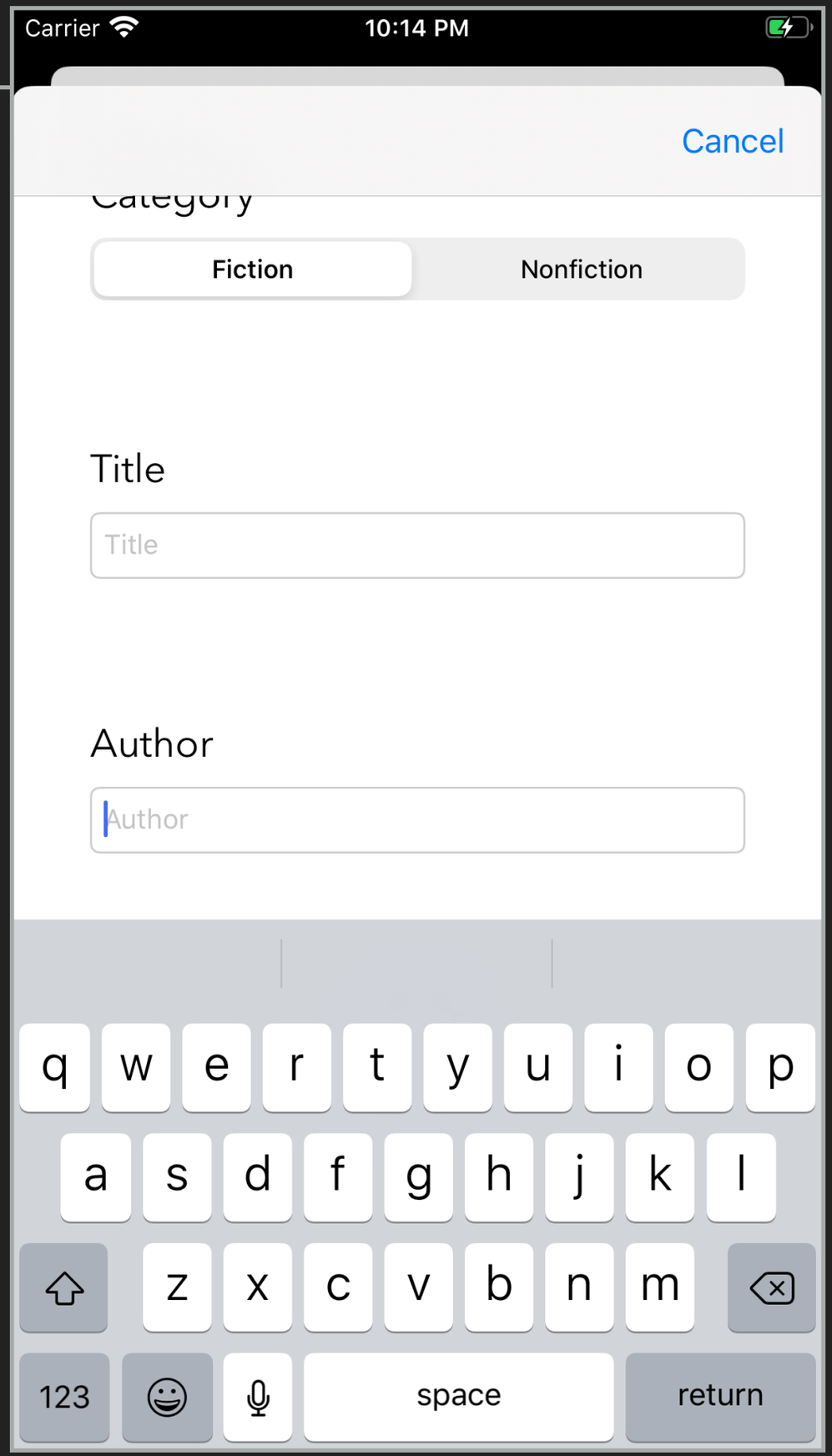
TEXT FIELD

- Option 1: Embed everything in a UIScrollView and adjust the scroll view when the keyboard appears
- Note - this can be pretty tricky!



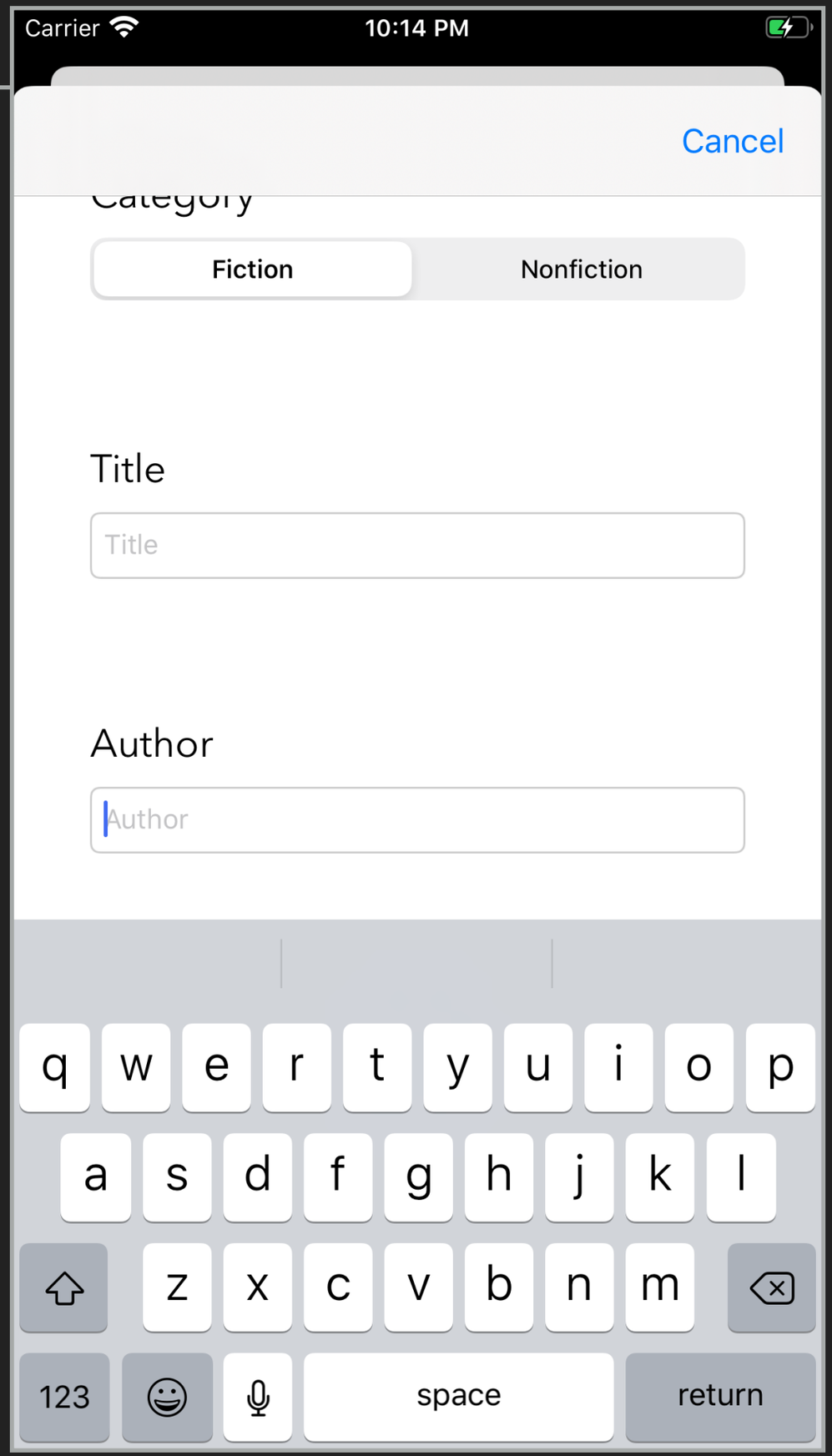
TEXT FIELD

- Option 2: Use a UITableViewController
- Text fields will scroll automatically! 🎉



TEXT FIELD

- Option 2: Use a UITableViewController
- Text fields will scroll automatically! 🎉



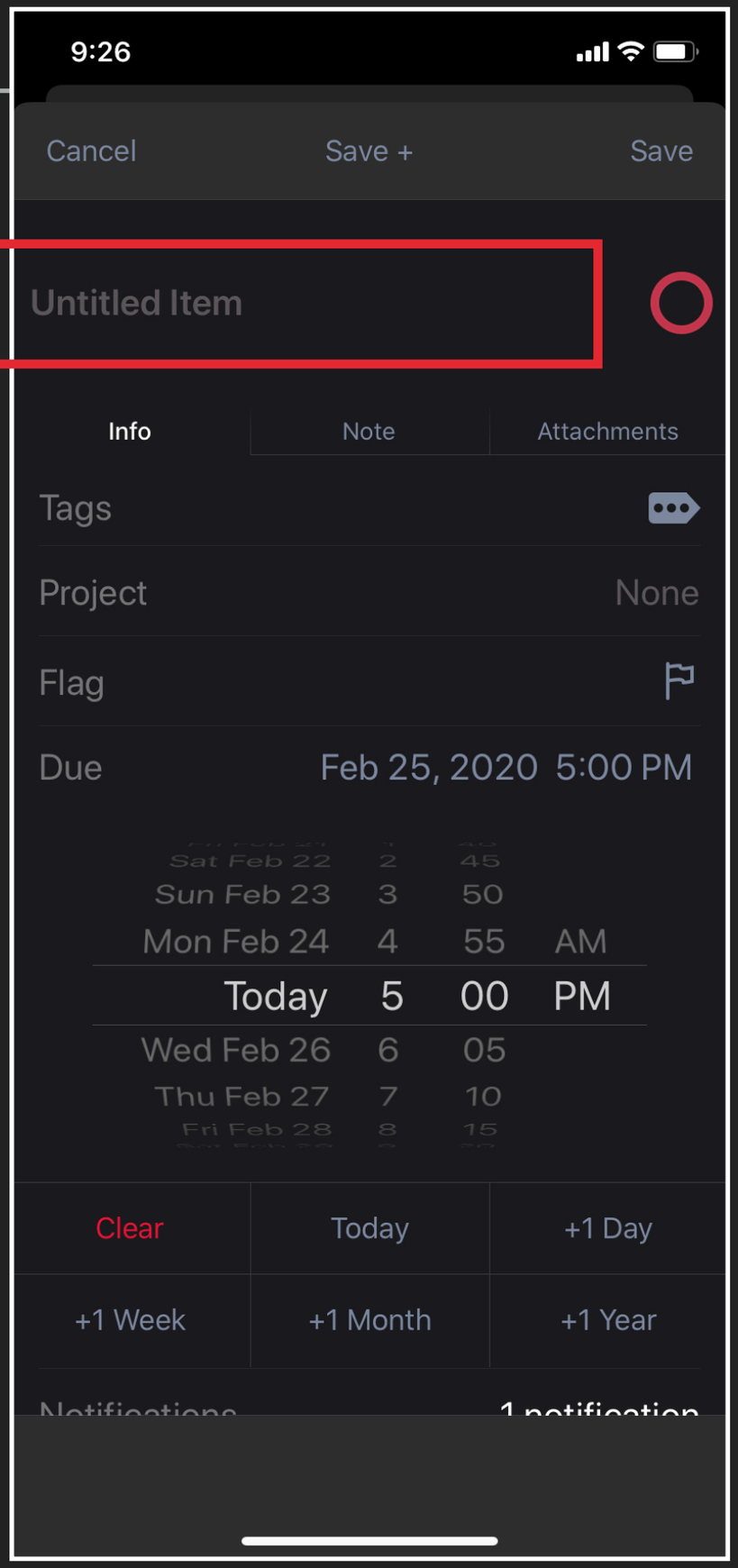
TEXT FIELD

- Since most people don’t *love* typing on mobile devices, here are a few more things you can do to make the experience as smooth as possible…

TEXT FIELD

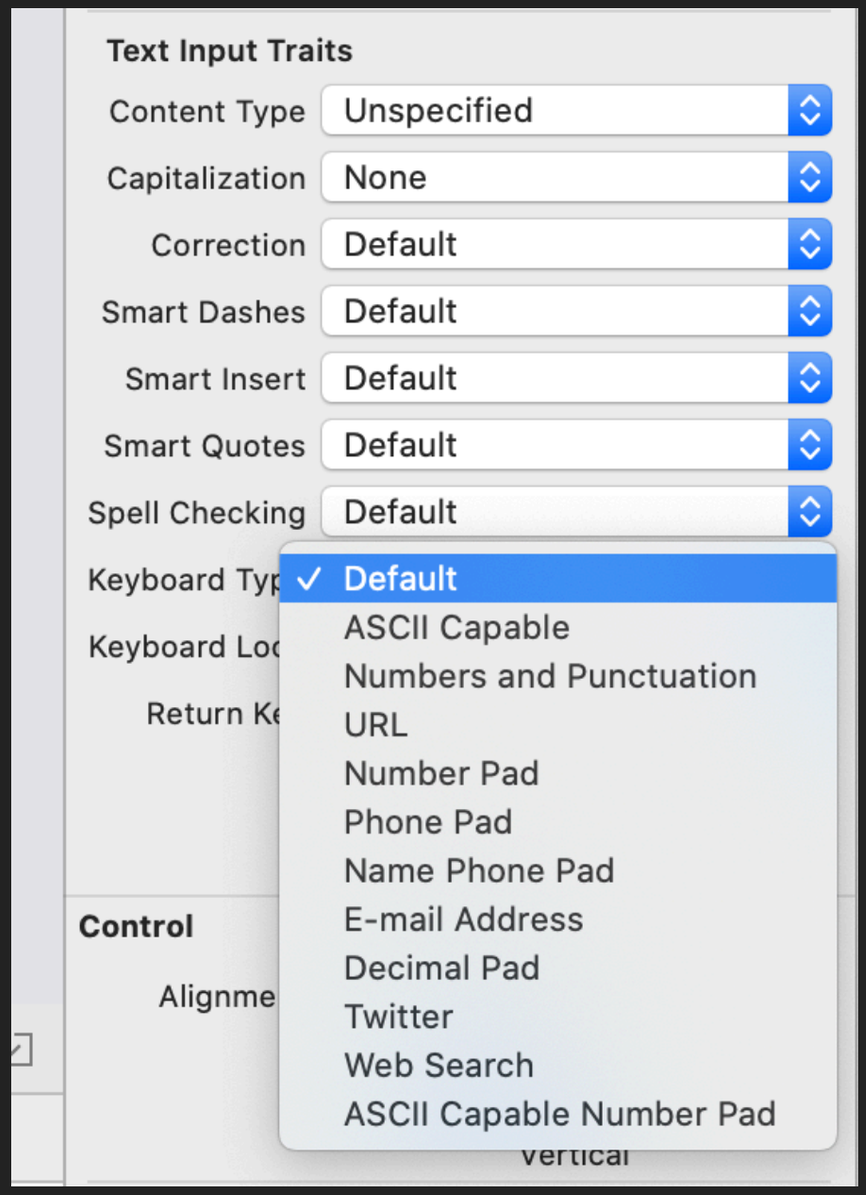
- Display placeholder text in the text field
  - Different font color
  - Disappears when the user starts typing
- Provides context about the purpose of the field

OmniFocus



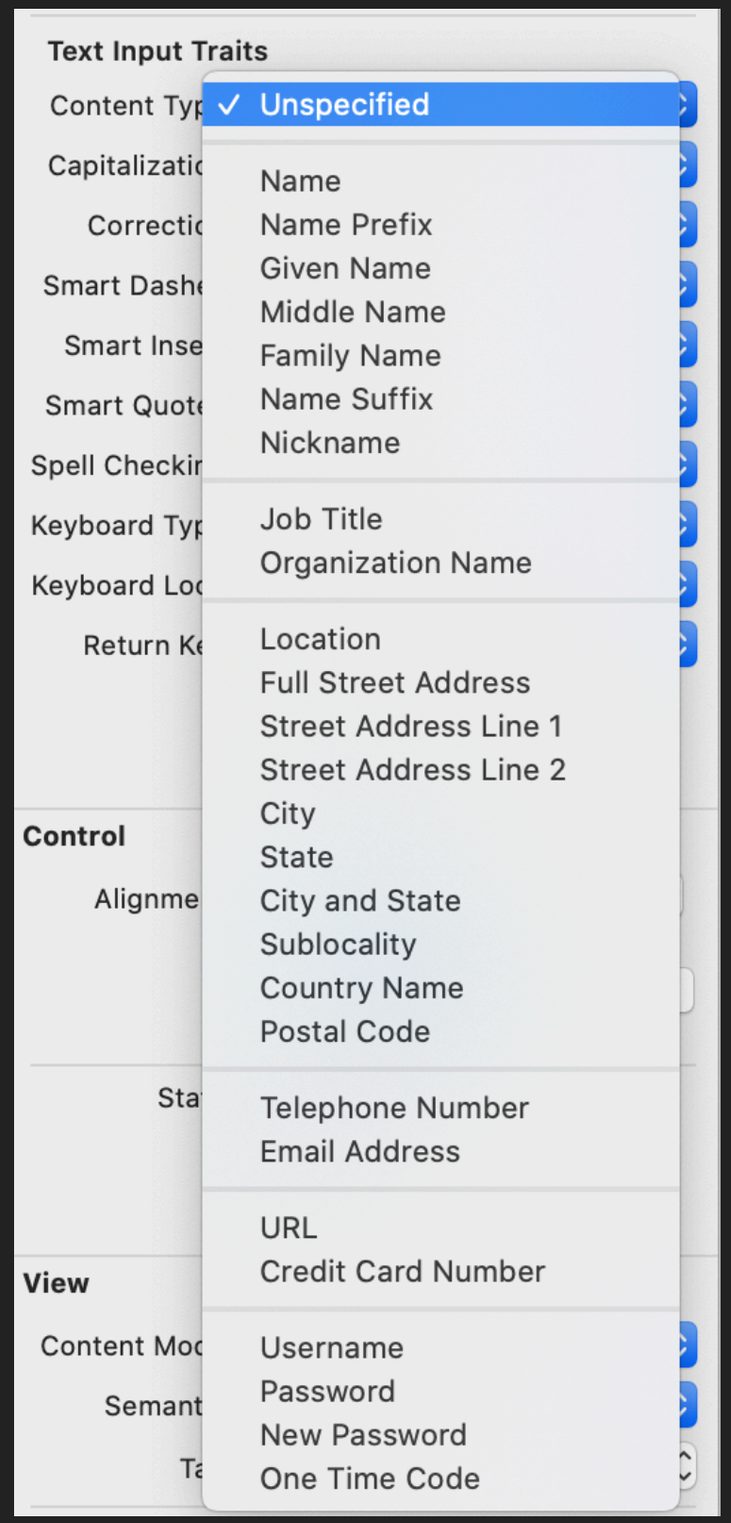
TEXT FIELD

- Use the correct keyboard type
- (Don’t show letters when the field only allows numbers)



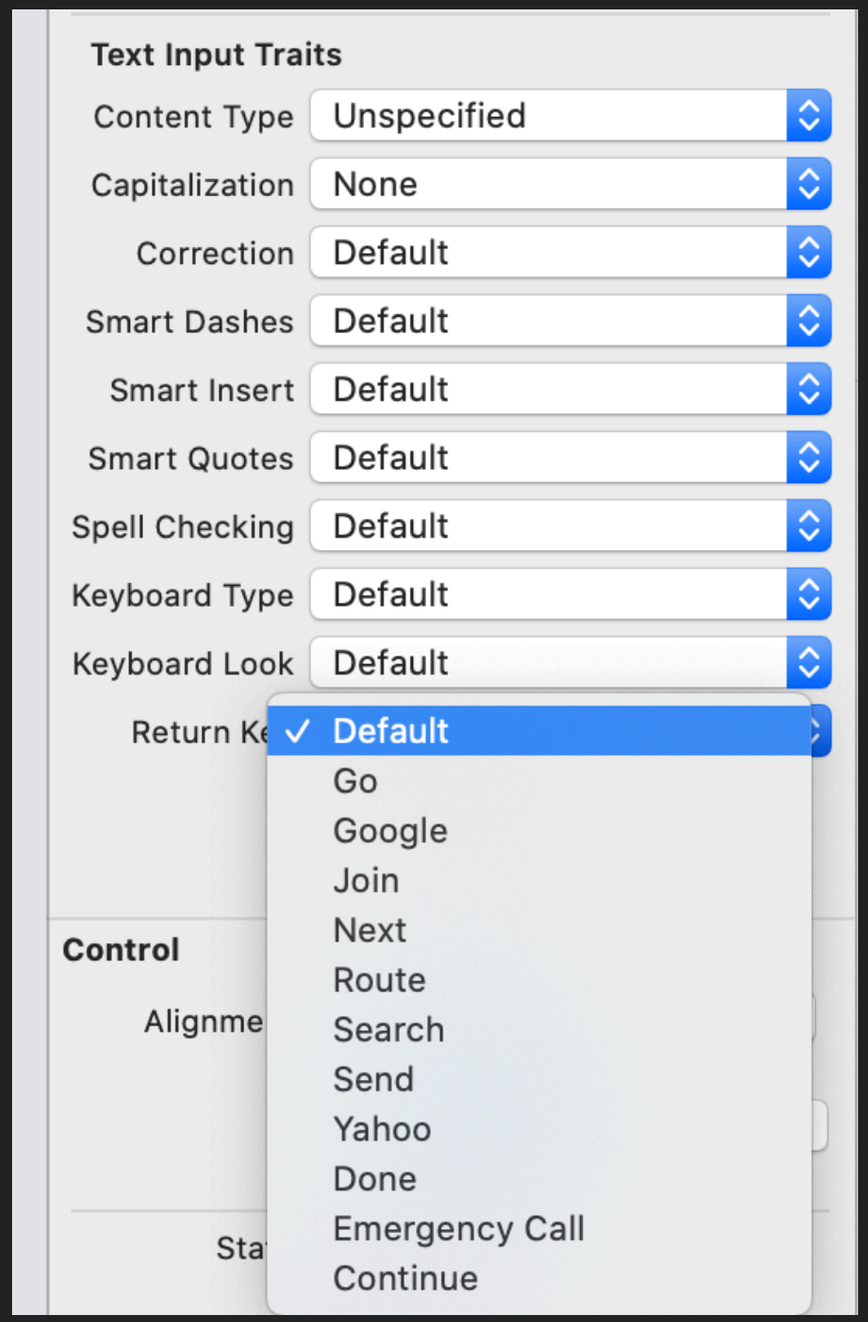
TEXT FIELD

- Set the content type whenever possible
- The keyboard will show suggestions, allowing users to quickly fill in basic info



TEXT FIELD

- Show the appropriate text on the return key
- If you have multiple text fields, automatically move the user to the next one



TEXT FIELD

- Most importantly, don’t allow your users to get “stuck”
  - Make it easy to dismiss the keyboard
  - Scroll text fields up so that they’re above the keyboard

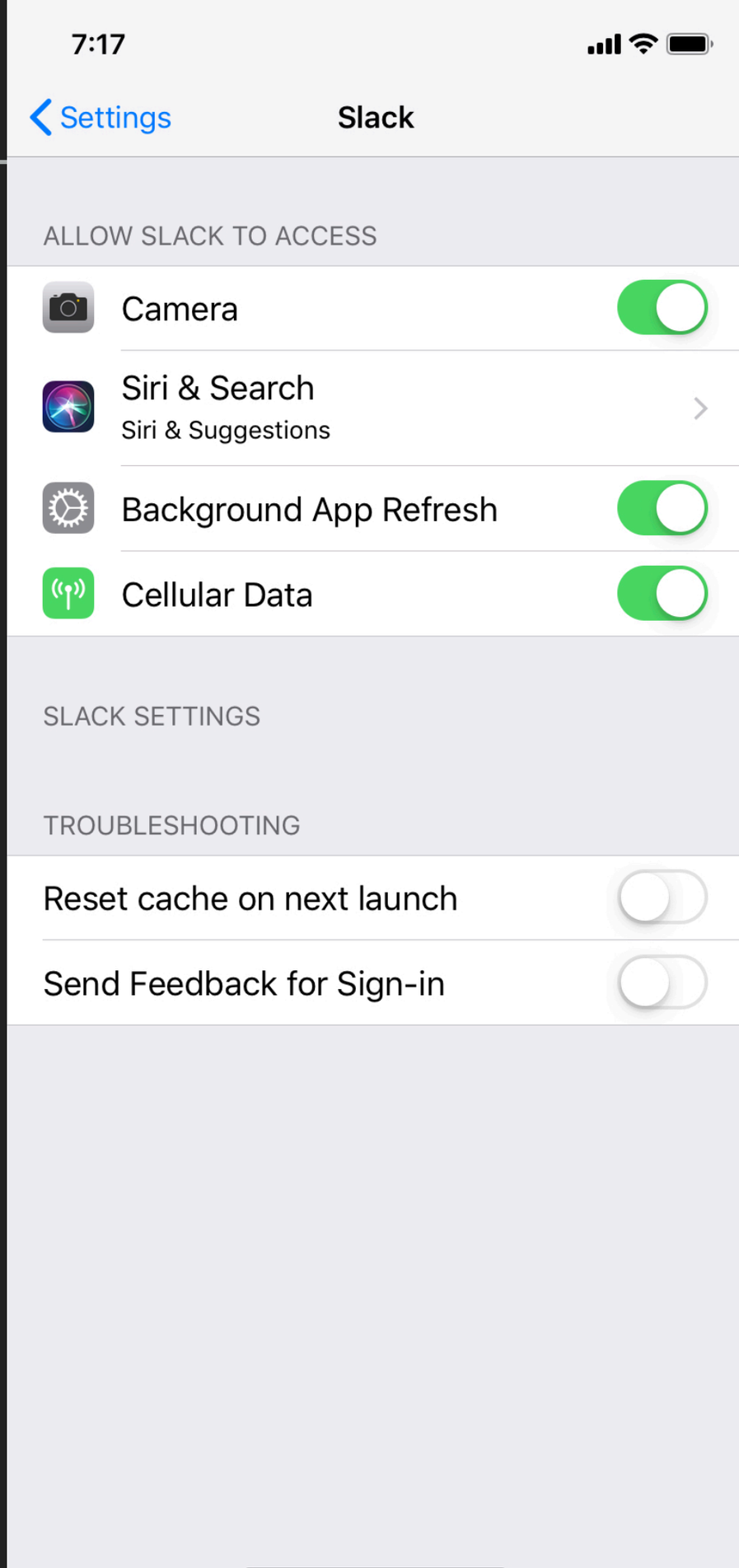
CREATING A

SETTINGS BUNDLE

SETTINGS APP

- Your app can have its own section in the Settings App

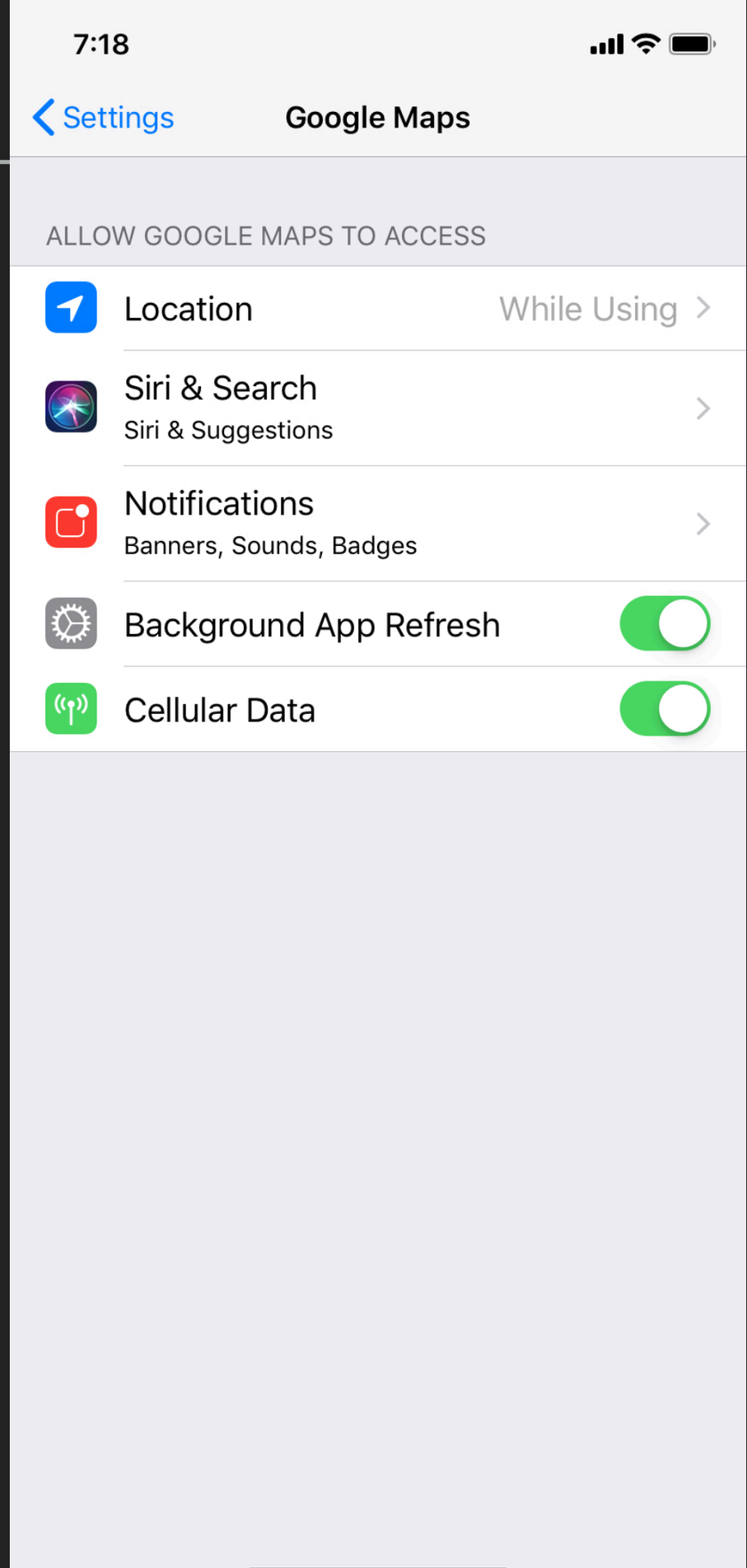
Slack



SETTINGS APP

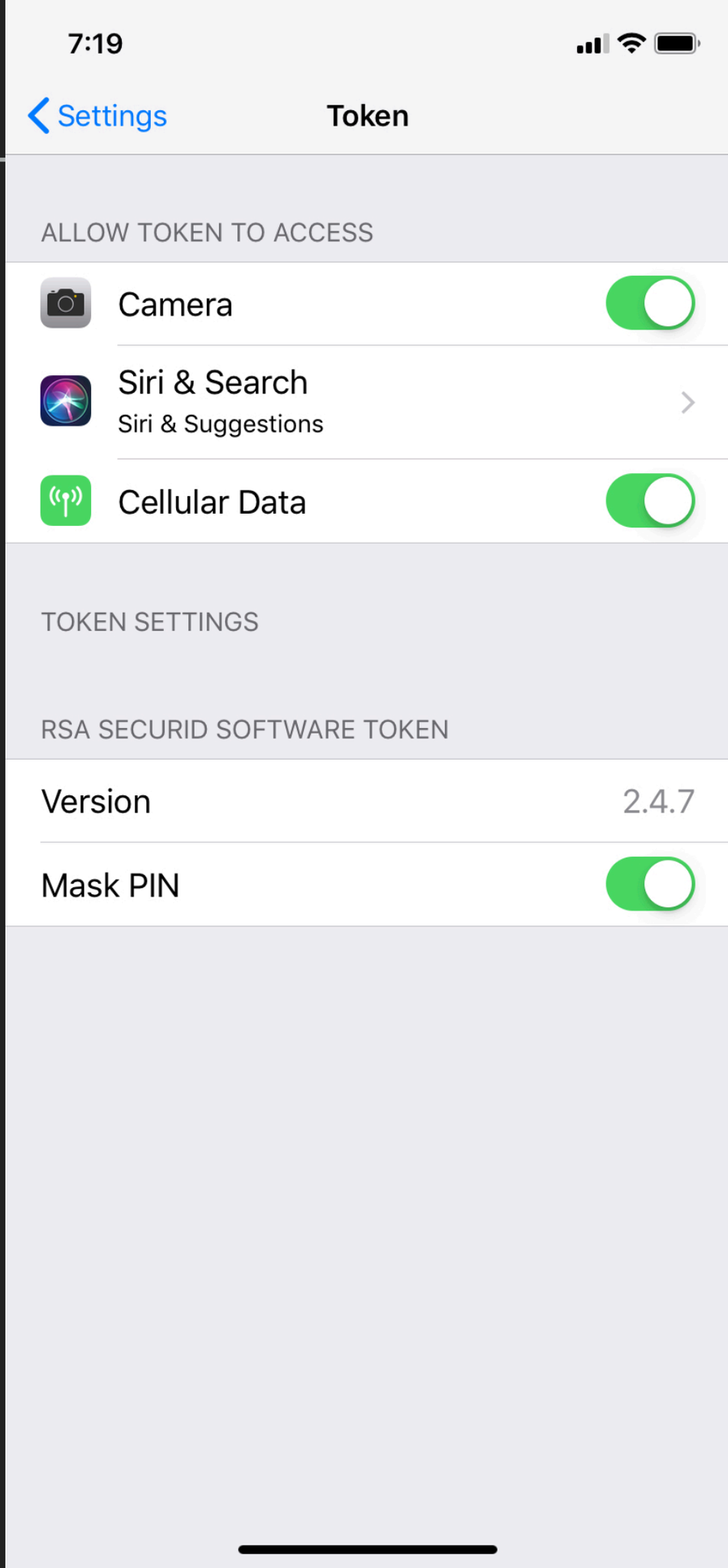
- This is where the user can change your app's access to things like:
  - Location
  - Camera
  - Contacts
  - Notifications

Google Maps



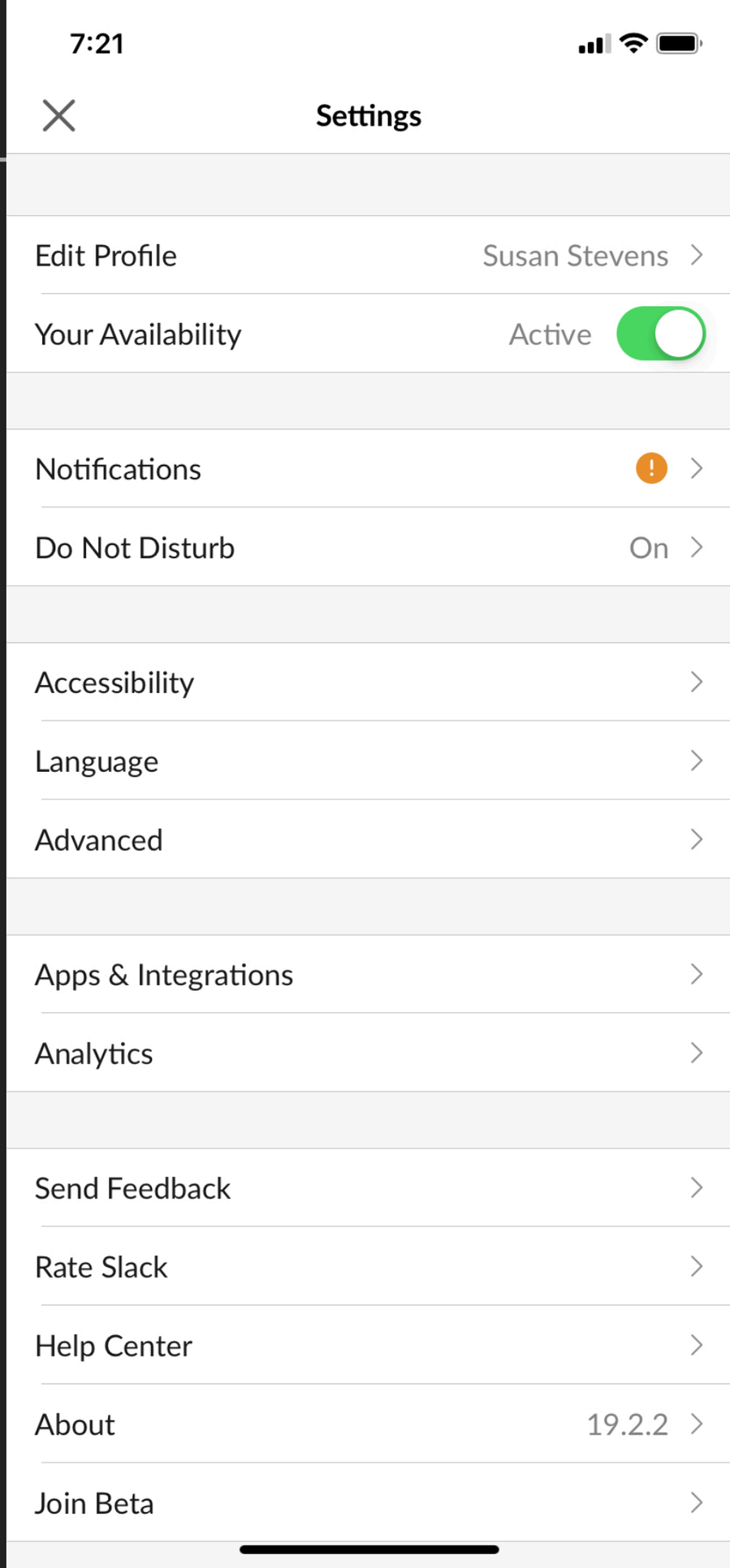
SETTINGS APP

- You can also include settings specific to your app
  - You do this by creating a Settings Bundle



SETTINGS APP

- Of course, you could also choose to place custom settings in the app itself



SETTINGS APP

- Pros of using a Settings Bundle
  - You don’t have to build a whole new screen
  - It eliminates potential clutter from your app

SETTINGS APP

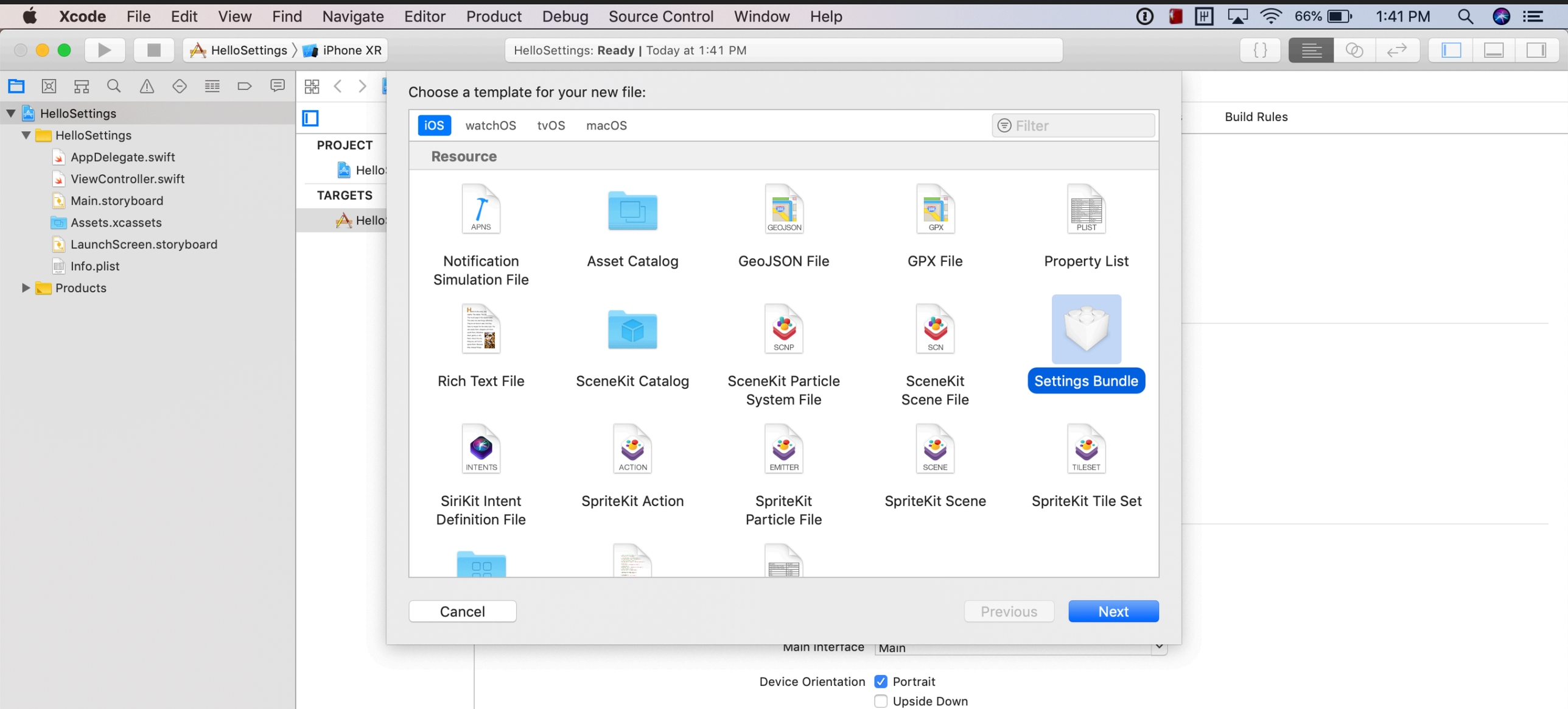
- Cons of using a Settings Bundle
  - Users may not know that it’s there
  - You get a limited number of controls

SETTINGS APP

- Secret use-case for a Settings Bundle: developing and testing your app
    - Point to different backend environments
    - Turn on and off caching
    - Show and hide features

TO ADD A SETTINGS BUNDLE

- File –> New –> File



EDITING YOUR SETTINGS BUNDLE

- Expand Settings.bundle and select Root.plist
- plist = Property List

HelloSettings 〉 iPhone XS | Running HelloSettings on iPhone XS

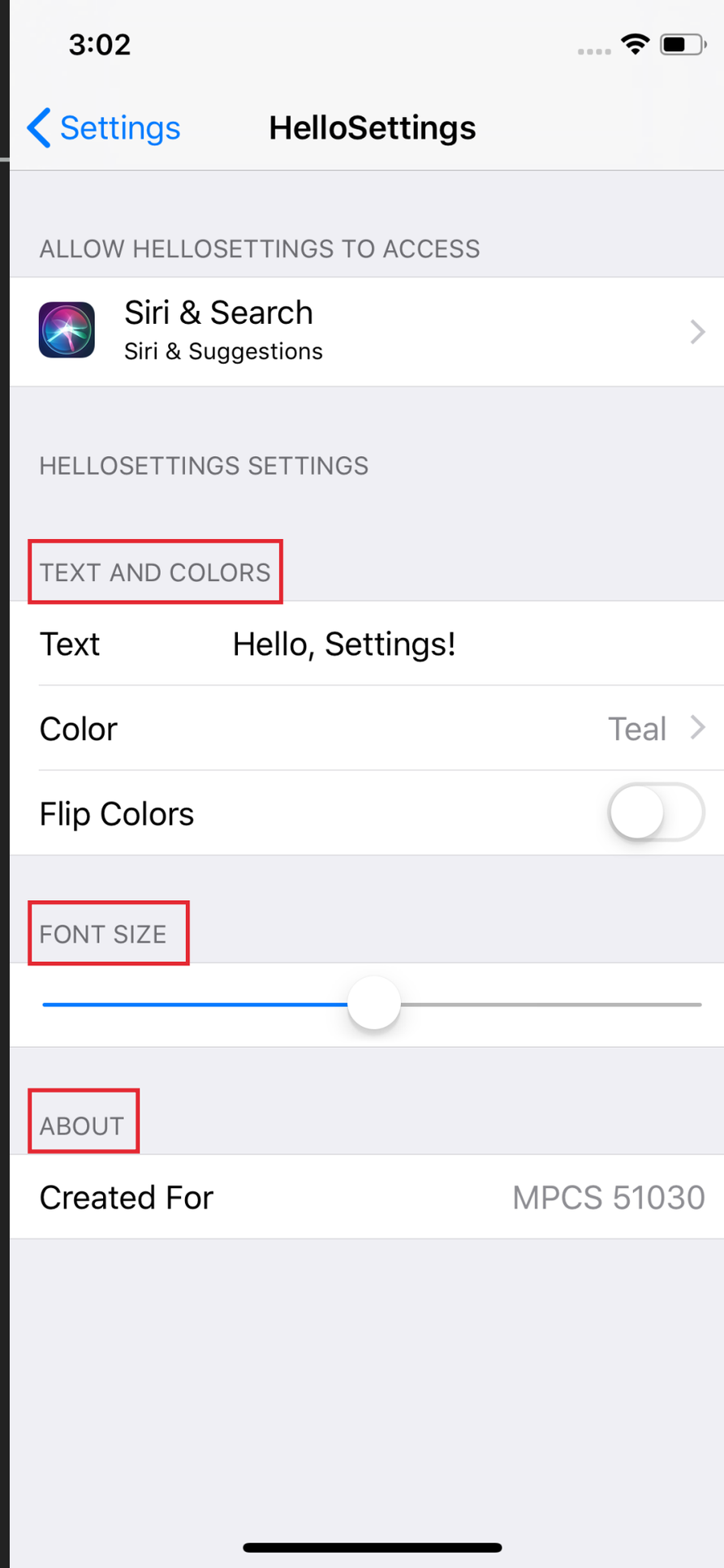
HelloSettings 〉 HelloSettings 〉 Settings.bundle 〉 Root.plist 〉 No Selection

- HelloSettings
  - HelloSettings
    - AppDelegate.swift
    - ViewController.swift
    - UserDefaultsKey.swift
    - Color.swift
    - Main.storyboard
    - Settings.bundle
      - en.lproj
      - Root.plist
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
  - Products

Key	Type	Value
iPhone Settings Schema	Dictionary	(2 items)
Strings Filename	String	Root
Preference Items	Array	(8 items)
Item 0 (Group - Text and Colors)	Dictionary	(2 items)
Type	String	Group
Title	String	Text and Colors
Item 1 (Text Field - Text)	Dictionary	(8 items)
Type	String	Text Field
Title	String	Text
Identifier	String	text
Default Value	String	Hello, Settings!
Text Field Is Secure	Boolean	NO
Keyboard Type	String	Alphabet
Autocapitalization Style	String	None
Autocorrection Style	String	No Autocorrection
Item 2 (Multi Value - Color)	Dictionary	(6 items)
Type	String	Multi Value
Title	String	Color
Identifier	String	colorId
Default Value	String	teal
Values	Array	(3 items)
Item 0	String	teal
Item 1	String	purple
Item 2	String	orange

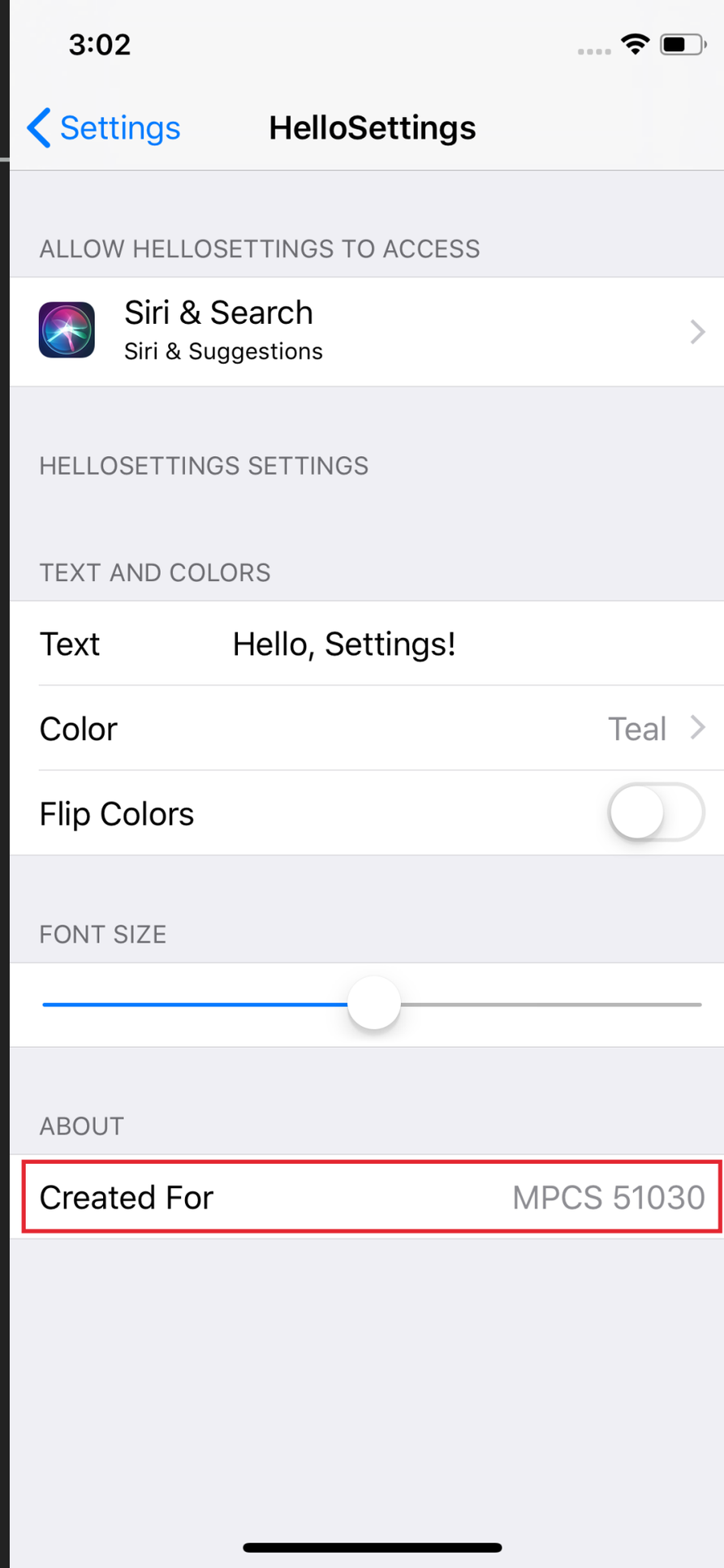
EDITING YOUR SETTINGS BUNDLE

- Under Preference Items, you can add:
  - Groups



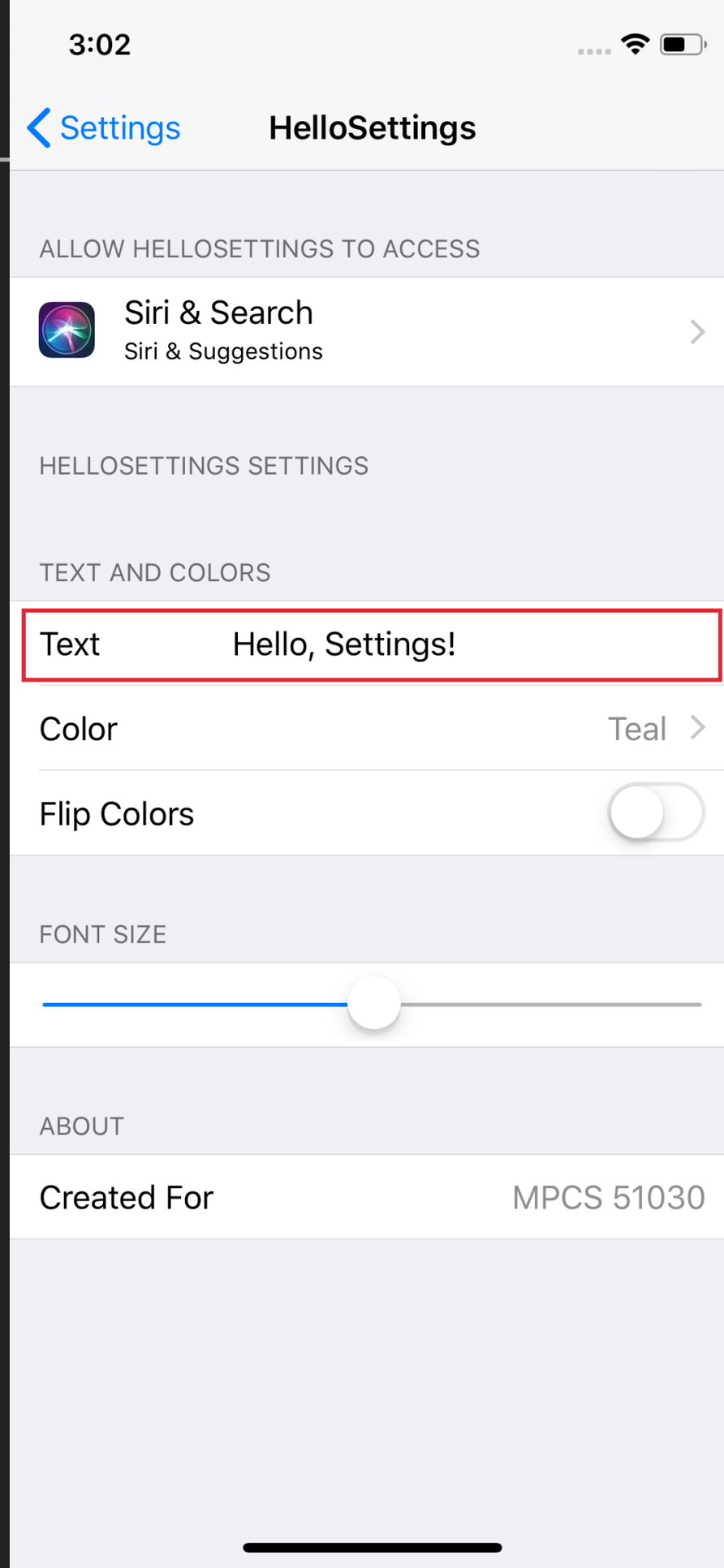
EDITING YOUR SETTINGS BUNDLE

- Under Preference Items, you can add:
  - Titles (not editable by the user)



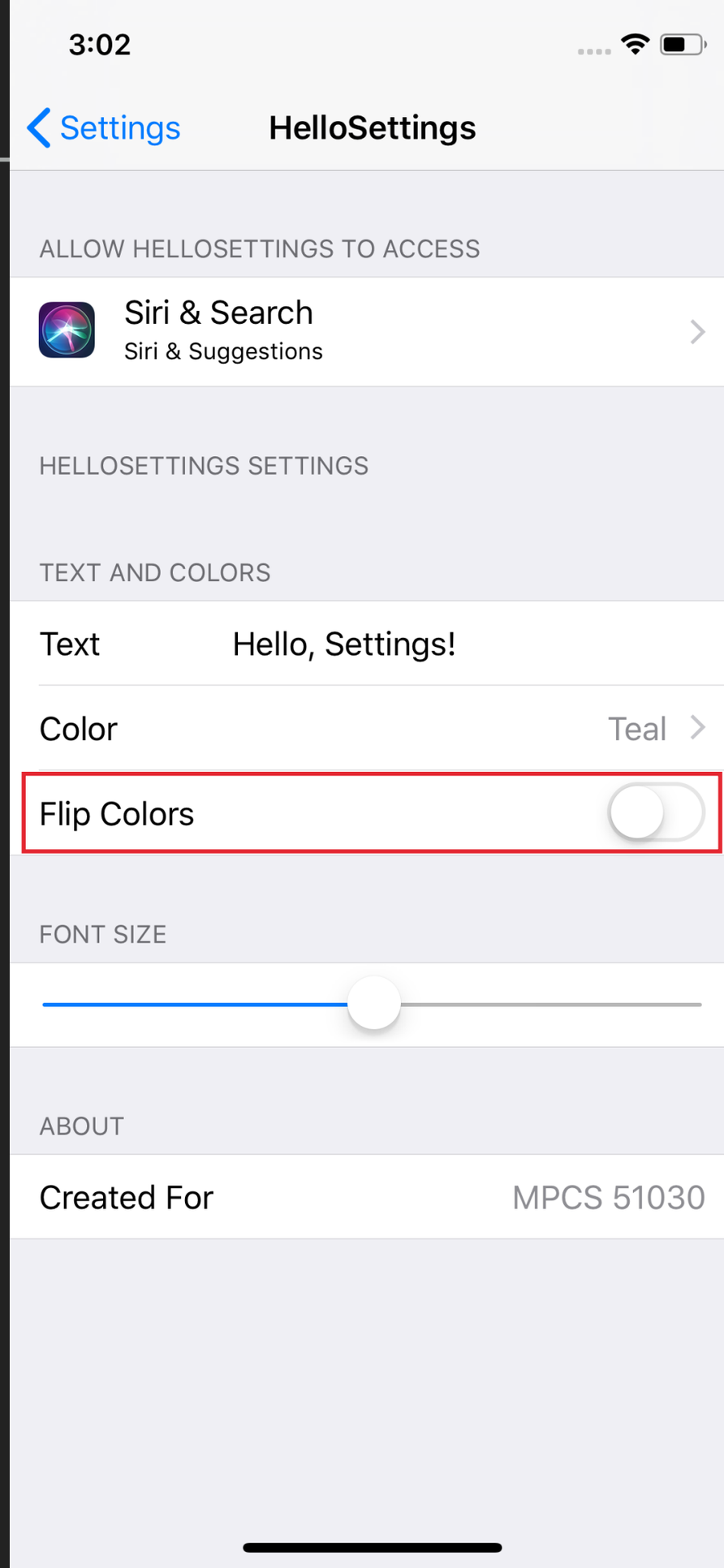
EDITING YOUR SETTINGS BUNDLE

- Under Preference Items, you can add:
  - Text Fields (editable by the user)



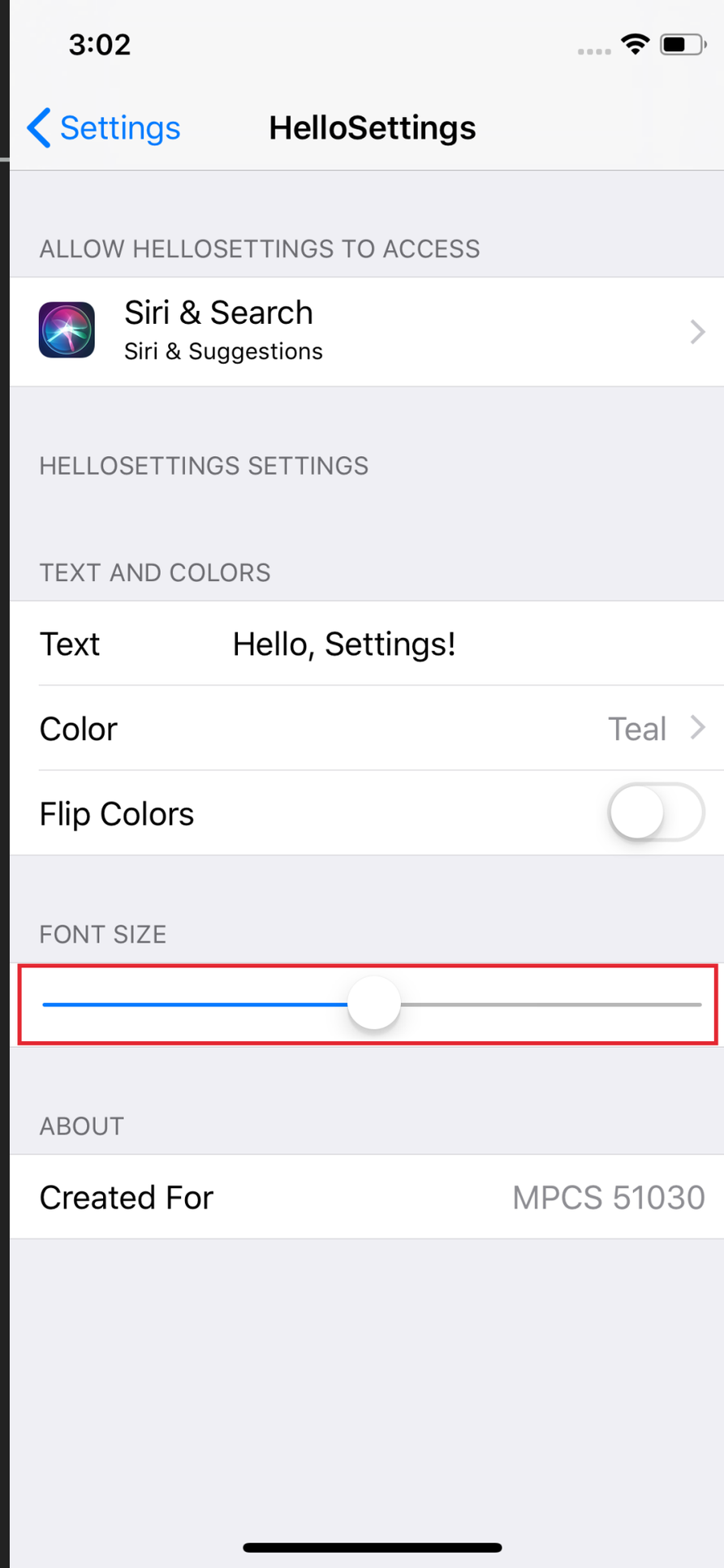
EDITING YOUR SETTINGS BUNDLE

- Under Preference Items, you can add:
  - Toggle Switches



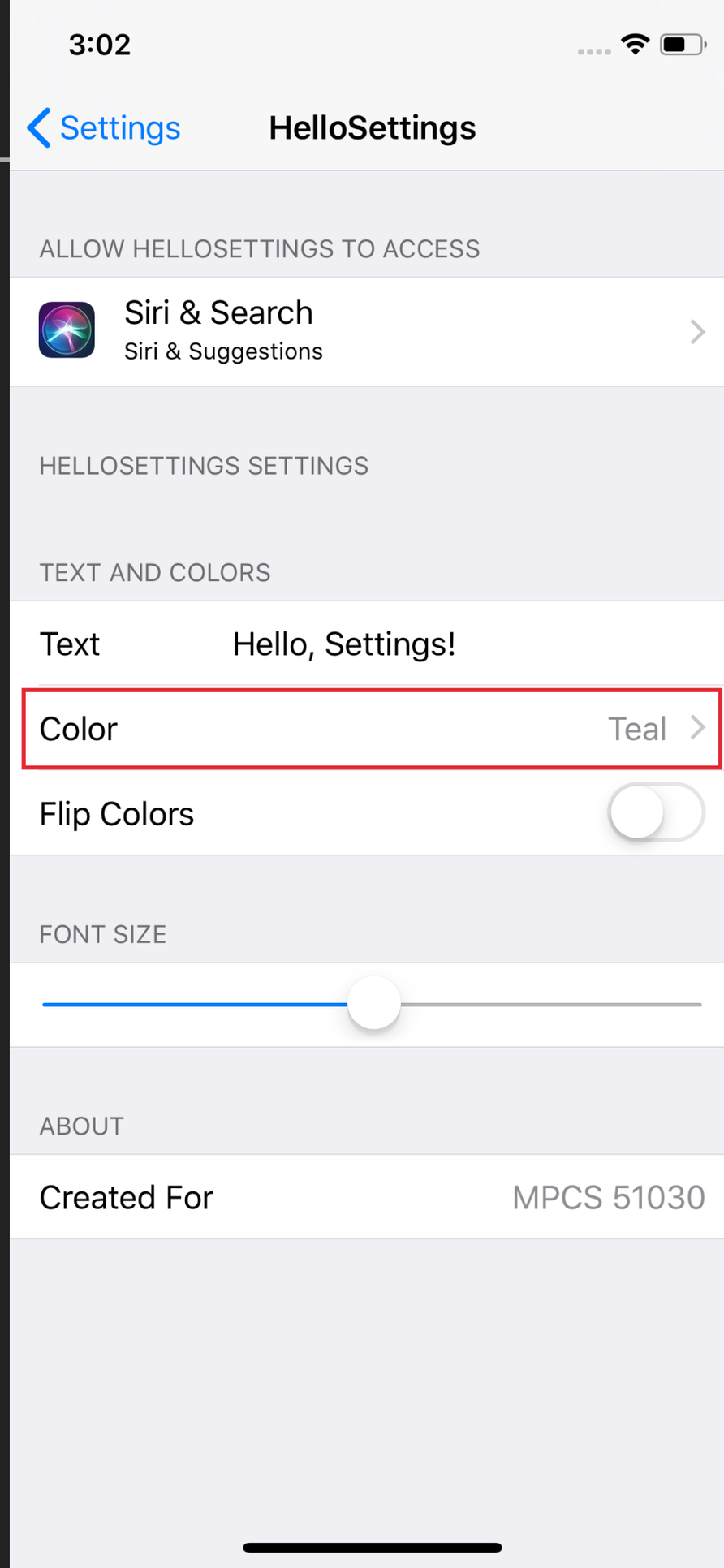
EDITING YOUR SETTINGS BUNDLE

- Under Preference Items, you can add:
  - Sliders



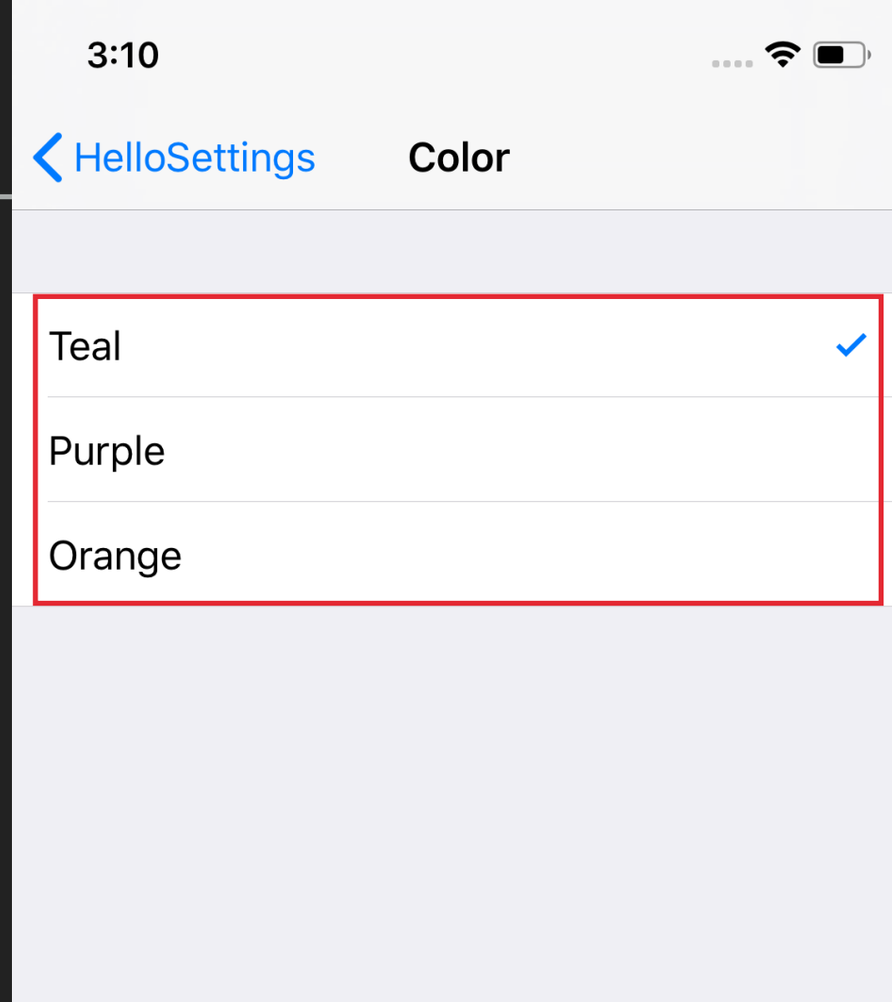
EDITING YOUR SETTINGS BUNDLE

- Under Preference Items, you can add:
  - Multi Values



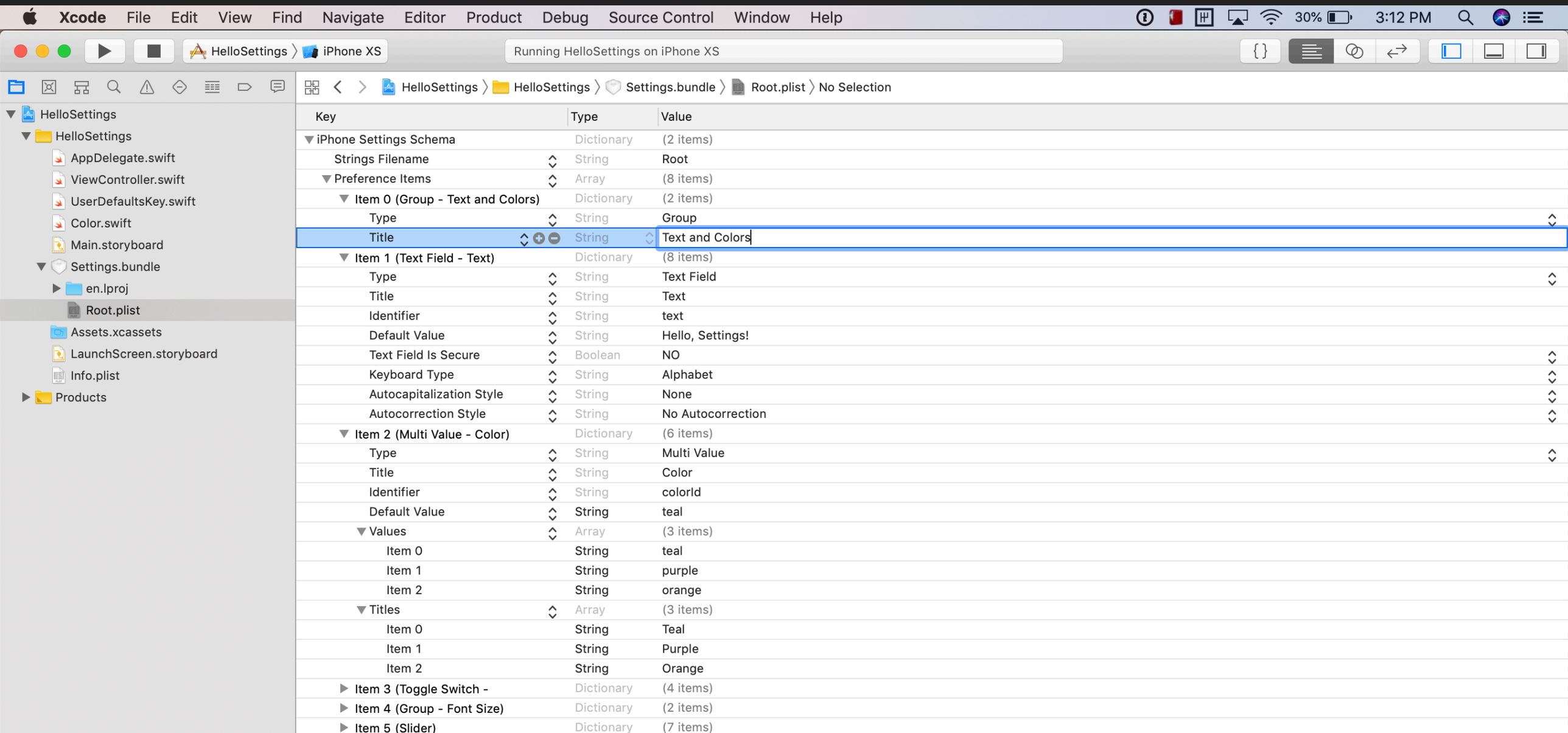
EDITING YOUR SETTINGS BUNDLE

- Under Preference Items, you can add:
  - Multi Values



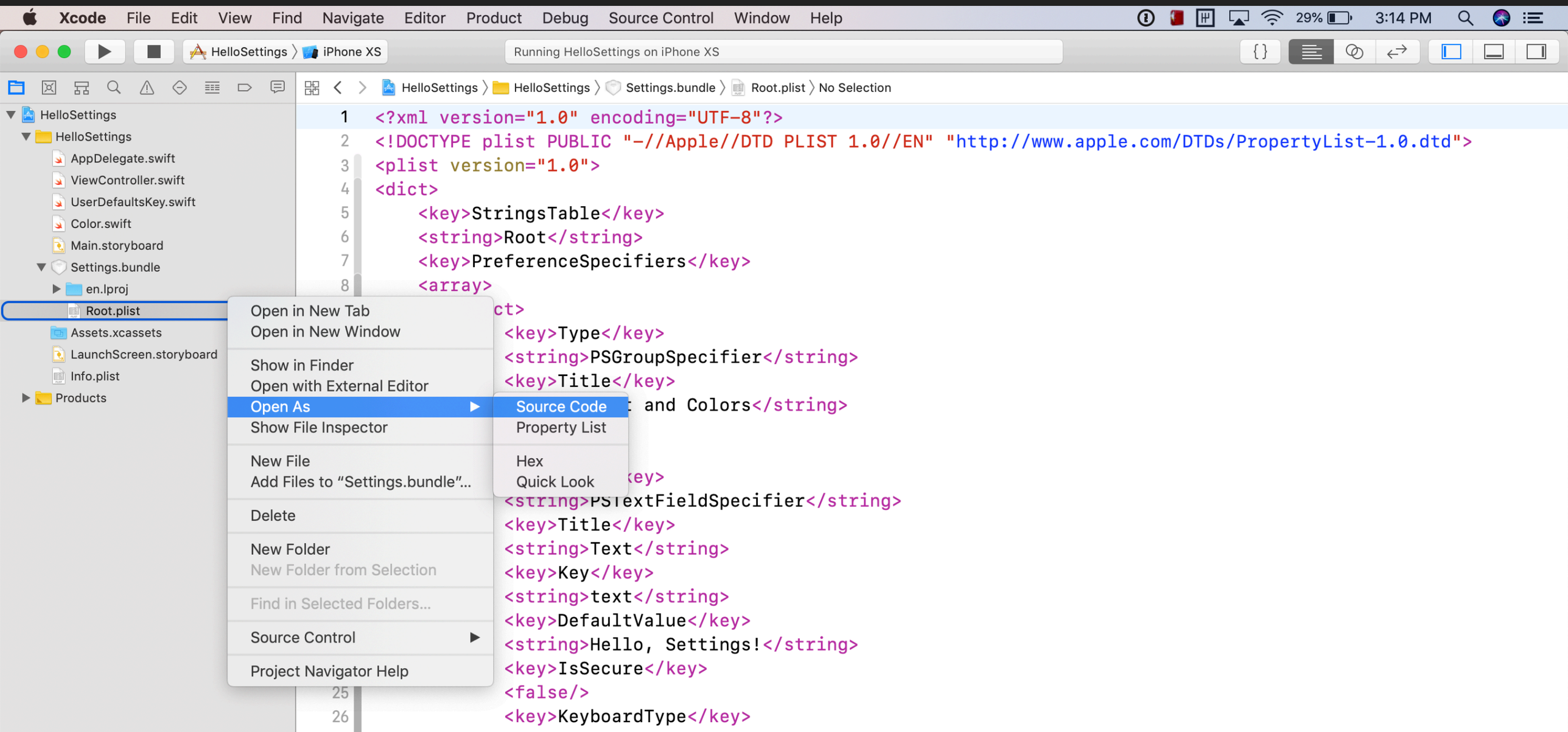
EDITING YOUR SETTINGS BUNDLE

- You can edit properties in the Property List view



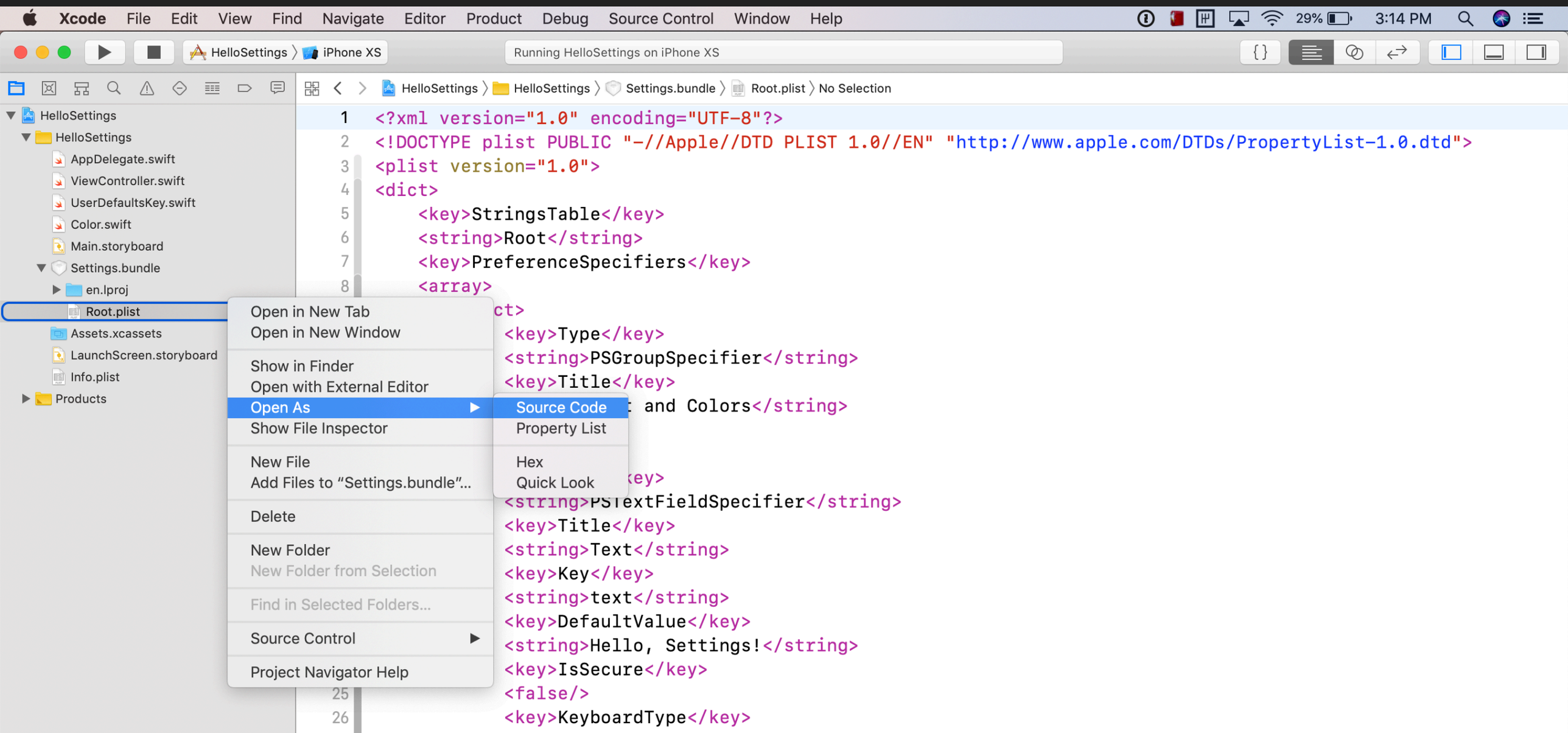
EDITING YOUR SETTINGS BUNDLE

- Or open as Source Code and edit the XML



EDITING YOUR SETTINGS BUNDLE

- Tip! It's easier to rearrange things in XML



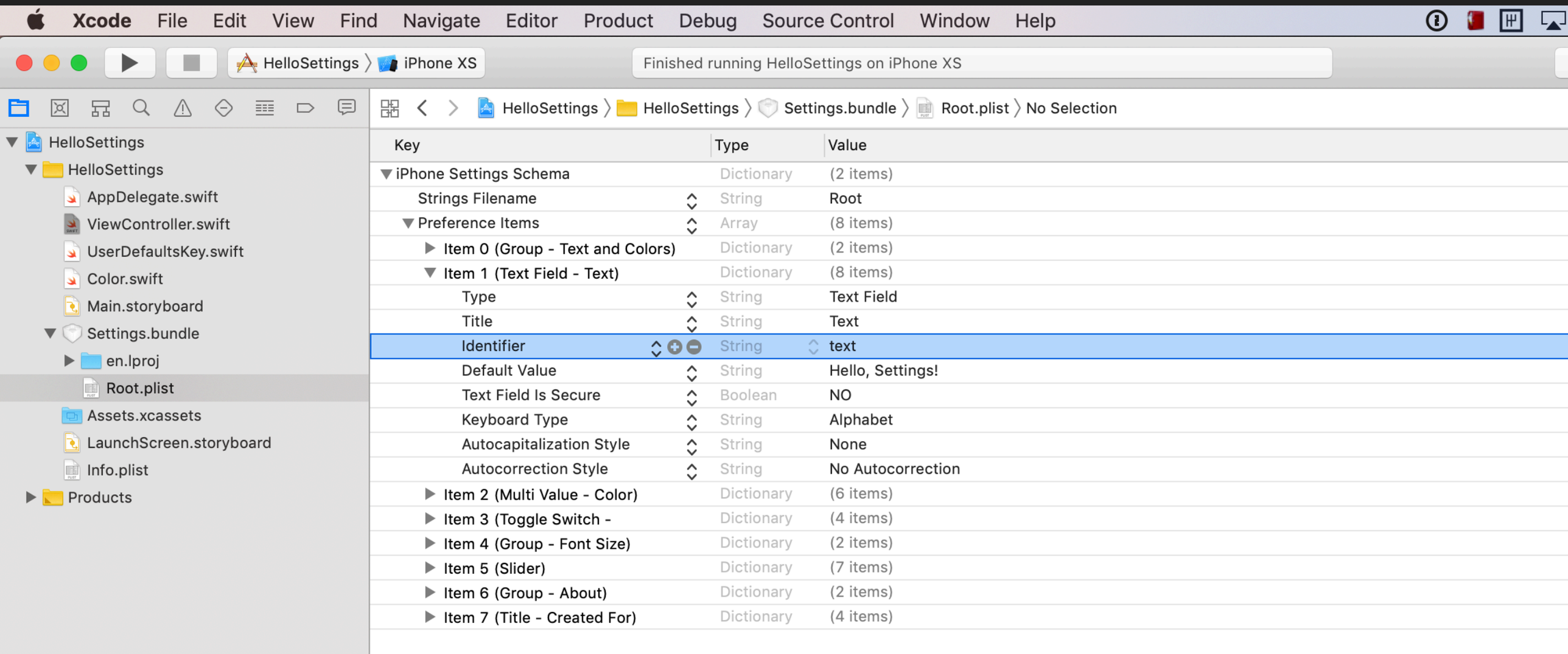
ACCESSING SETTINGS IN CODE

- Access your properties through UserDefaults

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

        let defaults = UserDefaults.standard

        let text = defaults.string(forKey: UserDefaultsKey.text.rawValue)
        let shouldFlipColors = defaults.bool(forKey: UserDefaultsKey.shouldFlipColors.rawValue)
        let fontSize = defaults.double(forKey: UserDefaultsKey.fontSize.rawValue)
        let colorId = defaults.string(forKey: UserDefaultsKey.colorId.rawValue)

        label.text = text
        label.font = UIFont.systemFont(ofSize: CGFloat(fontSize), weight: .medium)

        let selectedColor = Color.from(colorId) ?? Color.teal
```

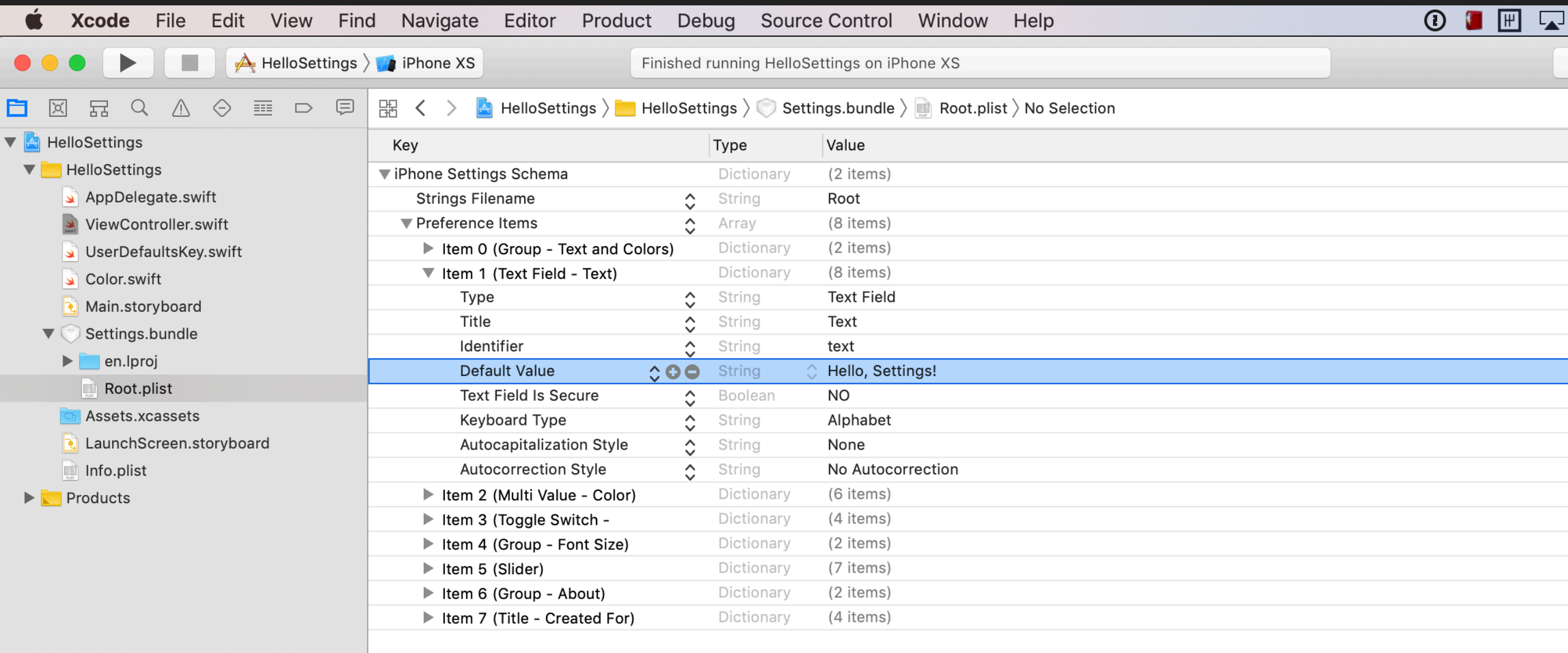
ACCESSING SETTINGS IN CODE

- Identifiers set in Root.plist are your keys for accessing the values in UserDefaults



PROBLEM WITH DEFAULT VALUES

- Root.plist allows you to set default values for editable fields



PROBLEM WITH DEFAULT VALUES

- The default values will be shown in the Settings App automatically
- However, UserDefaults will NOT be aware of those values

SOLUTION FOR DEFAULT VALUES

- Register defaults when the app launches

```swift
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        guard UserDefaults.standard.string(forKey: "text") == nil else { return true }

        let initialDefaults: [String: Any] = [
            UserDefaultsKey.text.rawValue : "Hello, Settings",
            UserDefaultsKey.shouldFlipColors.rawValue : false,
            UserDefaultsKey.fontSize.rawValue : 30.0,
            UserDefaultsKey.colorId.rawValue : "teal"
        ]

        UserDefaults.standard.register(defaults: initialDefaults)
        return true
    }
}
```

Check if nullable property already has a value

SOLUTION FOR DEFAULT VALUES

- Register defaults when the app launches

```swift
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        guard UserDefaults.standard.string(forKey: "text") == nil else { return true }

        let initialDefaults: [String: Any] = [
            UserDefaultsKey.text.rawValue : "Hello, Settings",
            UserDefaultsKey.shouldFlipColors.rawValue : false,
            UserDefaultsKey.fontSize.rawValue : 30.0,
            UserDefaultsKey.colorId.rawValue : "teal"
        ]

        UserDefaults.standard.register(defaults: initialDefaults)
        return true
    }
}
```

If not, register initial defaults with UserDefaults

OBSERVING CHANGES WITH

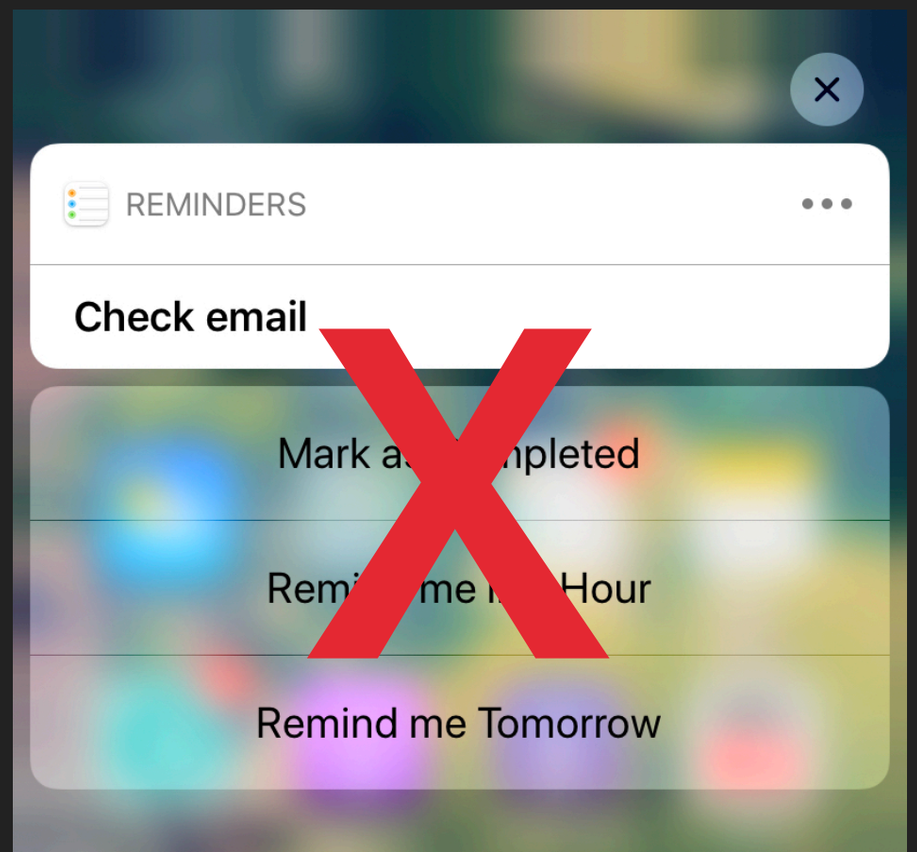
NOTIFICATION CENTER

OBSERVING CHANGES

- When the user changes a setting, we want to update the app immediately
- How do we know when a setting has changed?

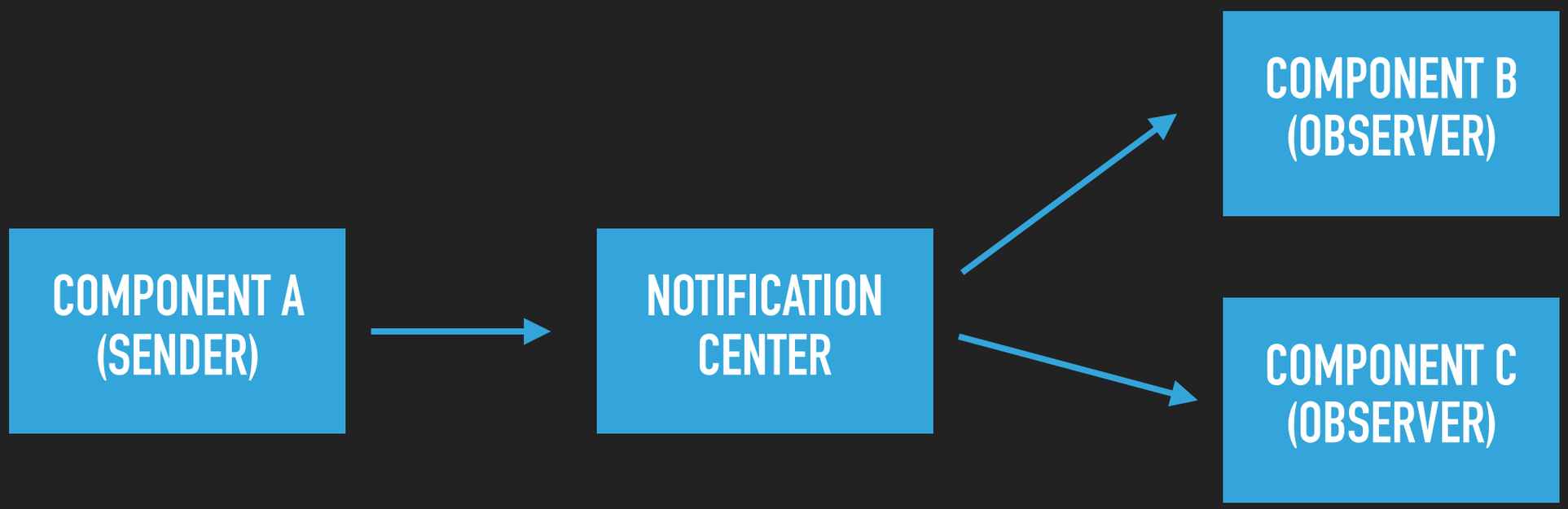
WHAT IS NOTIFICATION CENTER?

- Mechanism for broadcasting information to different parts of your app
- Note: This not related to the notifications shown to users



WHAT IS NOTIFICATION CENTER?

- A sender sends notifications to Notification Center
- Notification Center dispatches it to registered observers



POSTING A NOTIFICATION

- Notifications are identified by name
- `Object` is the sender of the notification

```
NotificationCenter.default.post(name: Notification.Name("didUpdateUsername"),
                                object: self)
```

OBSERVING A NOTIFICATION

- `selector` is the method to be called when the notification is received
- If `object` is nil, the observer will receive notifications from any sender

```
NotificationCenter.default.addObserver(
    self,
    selector: #selector(displayNewUsername),
    name: Notification.Name("didUpdateUsername"),
    object: nil)
```

OBSERVING NOTIFICATIONS FROM USER DEFAULTS

```
NotificationCenter.default.addObserver(
    self,
    selector: #selector(displaySettings),
    name: UserDefaults.didChangeNotification,
    object: nil)
```

OTHER USEFUL NOTIFICATIONS

- `UIApplication` sends notifications when:
  - App's state changes (e.g., enters background)
  - App's orientation changes
  - User takes a screenshot
  - ... and many more

OTHER USEFUL NOTIFICATIONS

- `AVAudioSession` sends notifications when:
  - Audio is interrupted (e.g., FaceTime call comes in)
  - Audio route changes (e.g., user unplugs headphones)

OTHER USEFUL NOTIFICATIONS

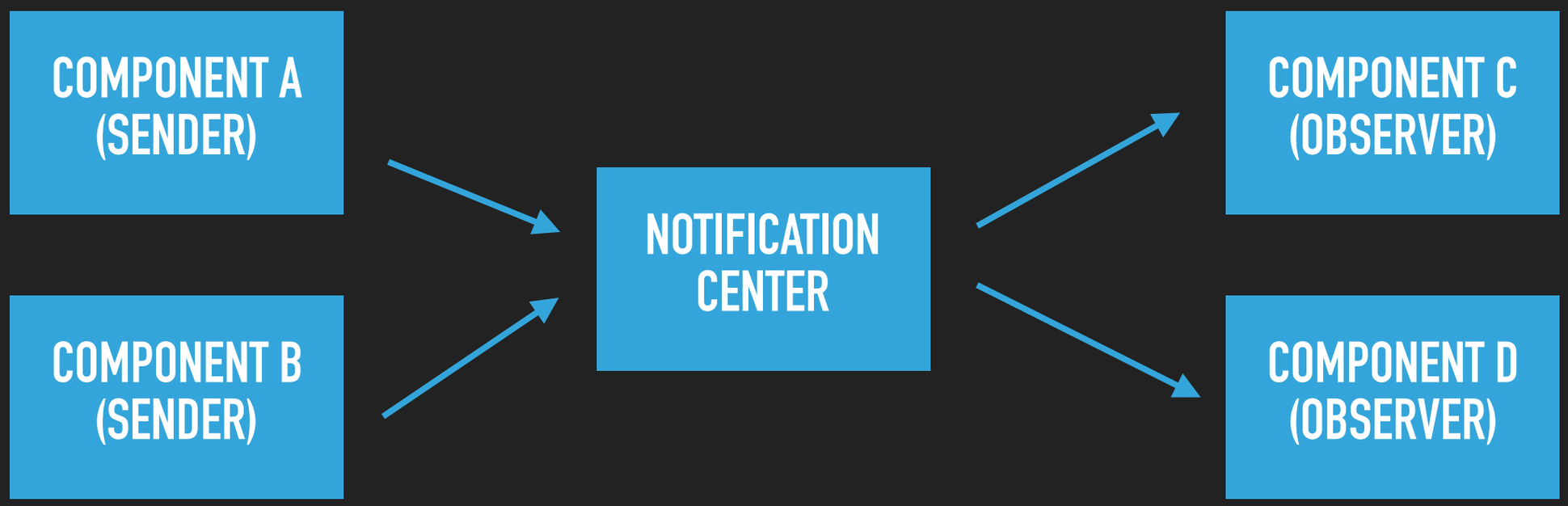
- `UIResponder` sends notifications:
    - Just before and after the keyboard is shown and hidden

DOWNSIDE OF NOTIFICATION CENTER

- One problem with Notification Center: control flow is difficult to follow

DOWNSIDE OF NOTIFICATION CENTER

- Example: When Component C receives a notification, how do we know where it came from?



RECOMMENDATIONS

- Use Notification Center only when there is no other good way to be notified of a change
- Use it primarily to observe events sent by Apple frameworks

RECOMMENDATIONS

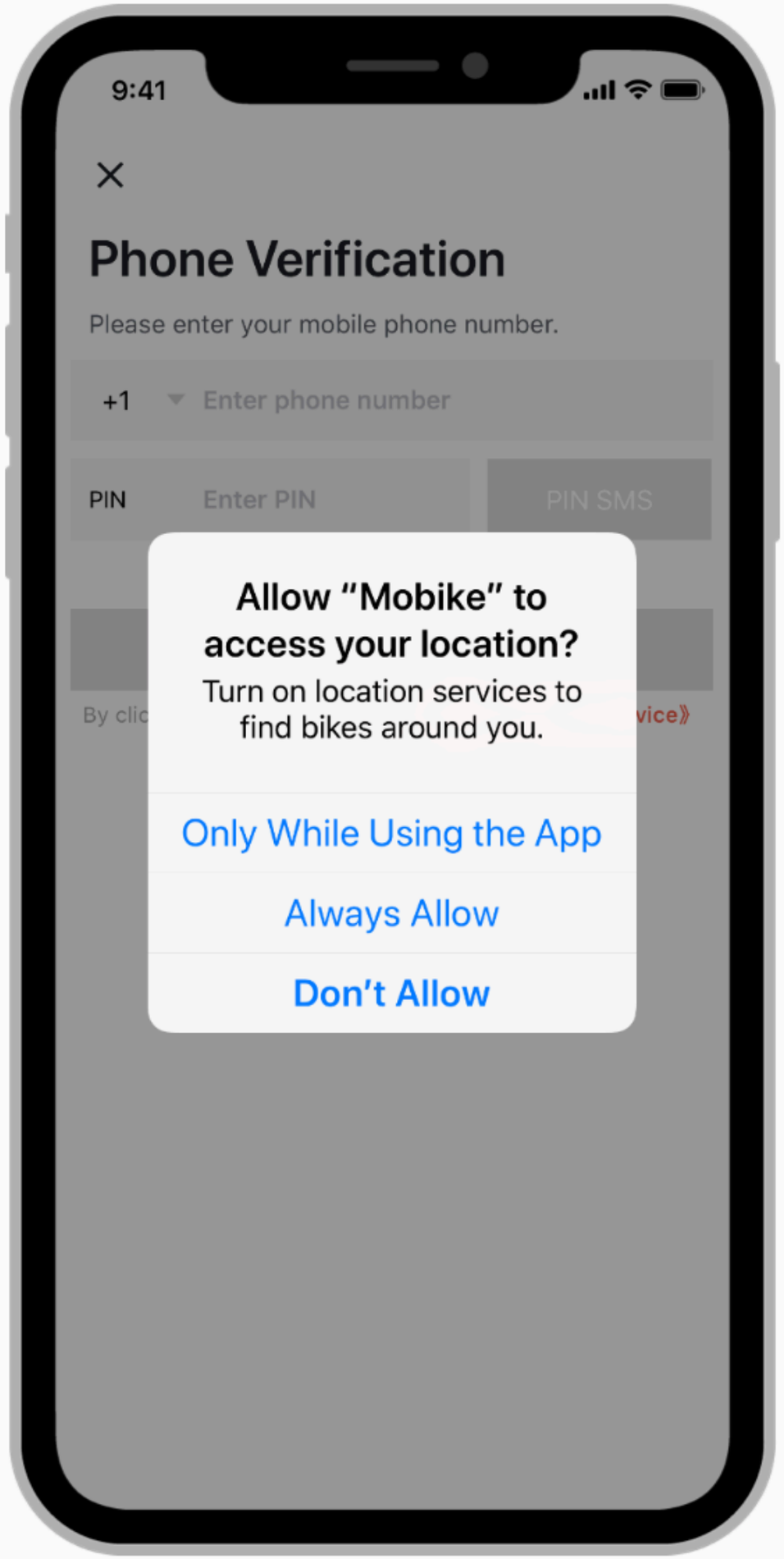
- Before sending your own notifications, consider other ways components can communicate, such as:
  - Delegates, closures, target-action

OPENING THE

SETTINGS APP

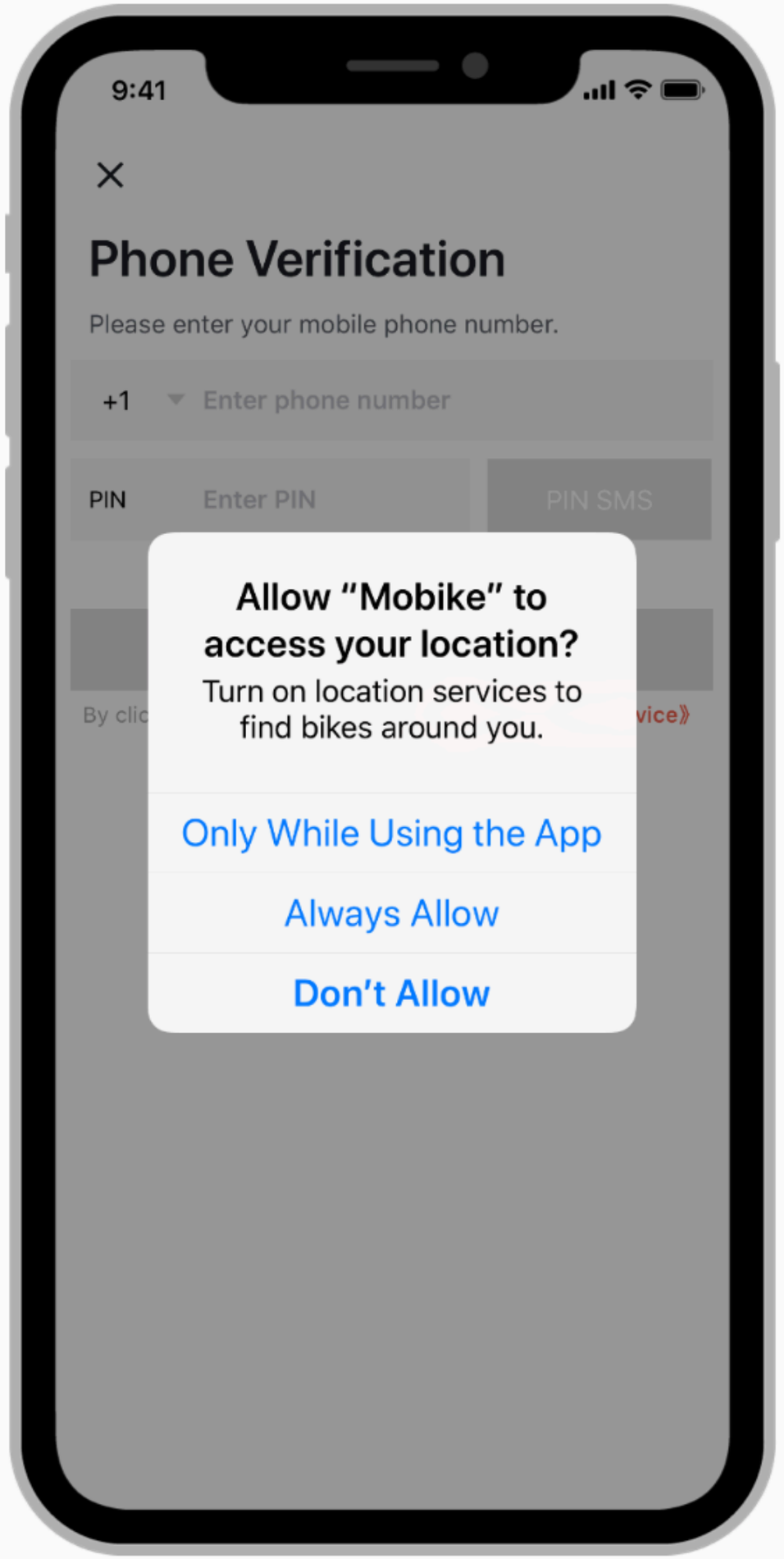
OPENING THE SETTINGS APP

- You may only show the system prompt requesting a particular type of access (location, camera, contacts, etc.) once



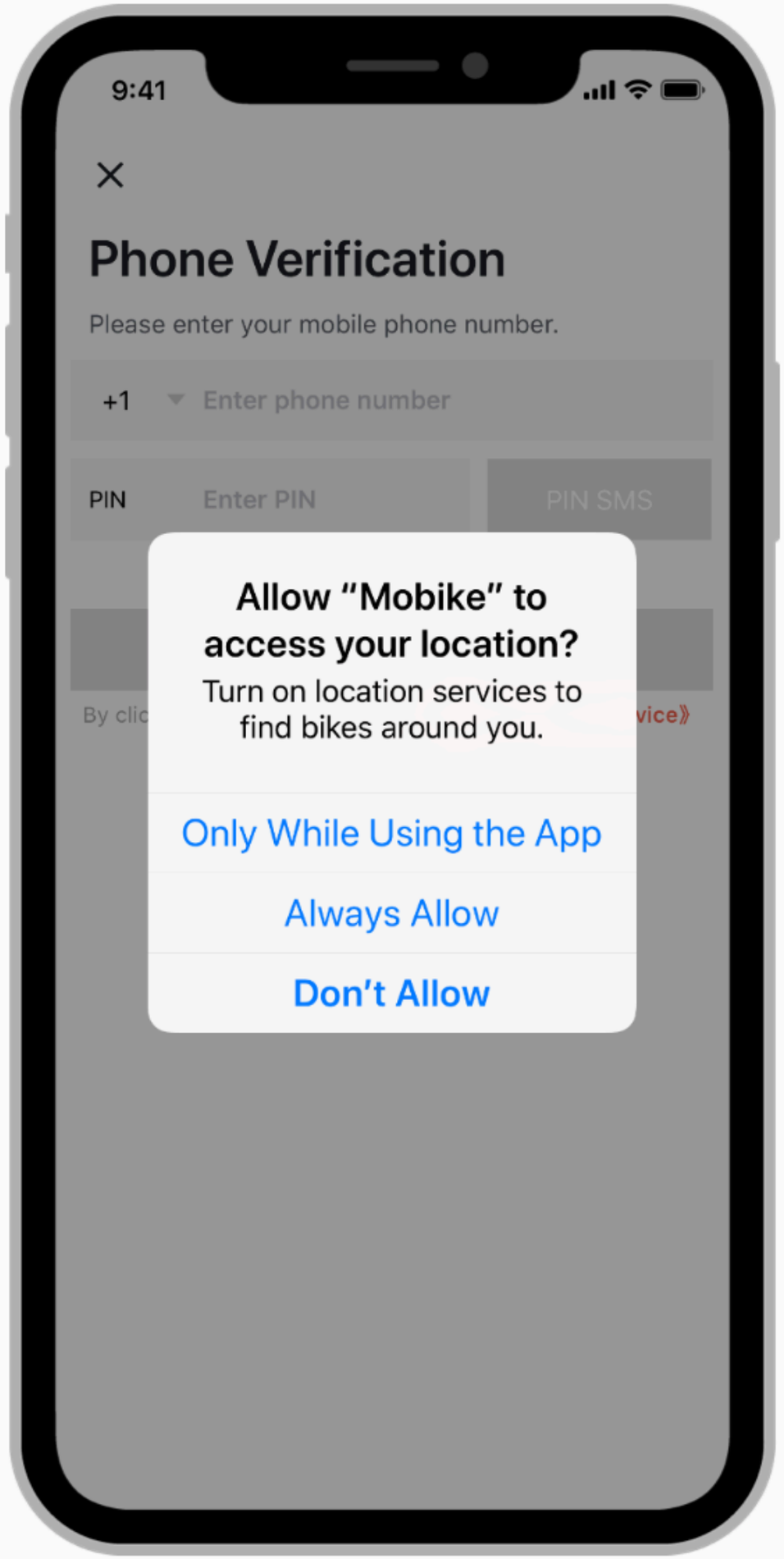
OPENING THE SETTINGS APP

- If the user denies access initially, the only way for them to grant it later is through the Settings App



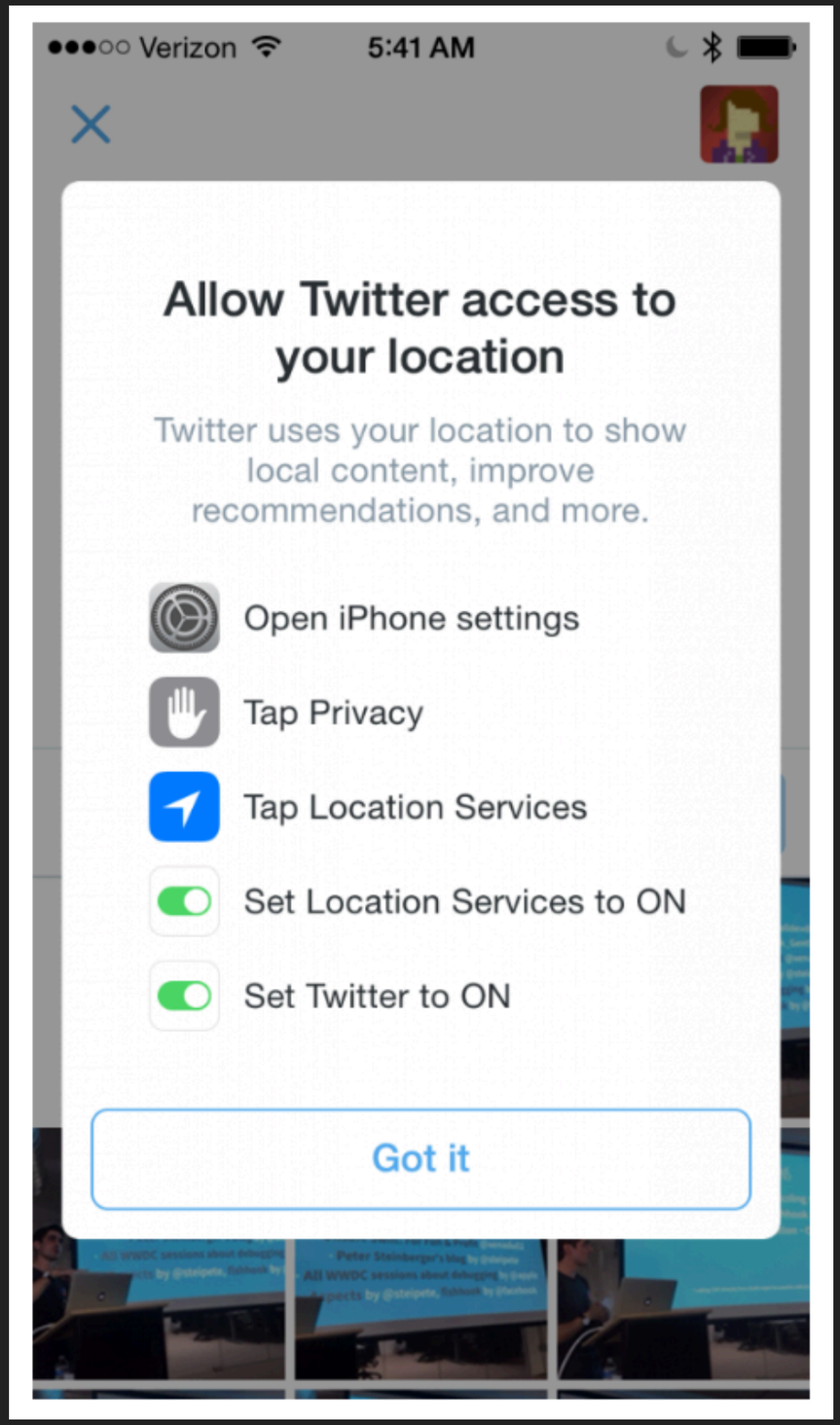
OPENING THE SETTINGS APP

- Why? Apple doesn’t want apps to bug users over and over again.



OLD METHOD

- Pre iOS 8: No way to automatically send user to the Settings App
  - Instead, apps provided an elaborate set of instructions



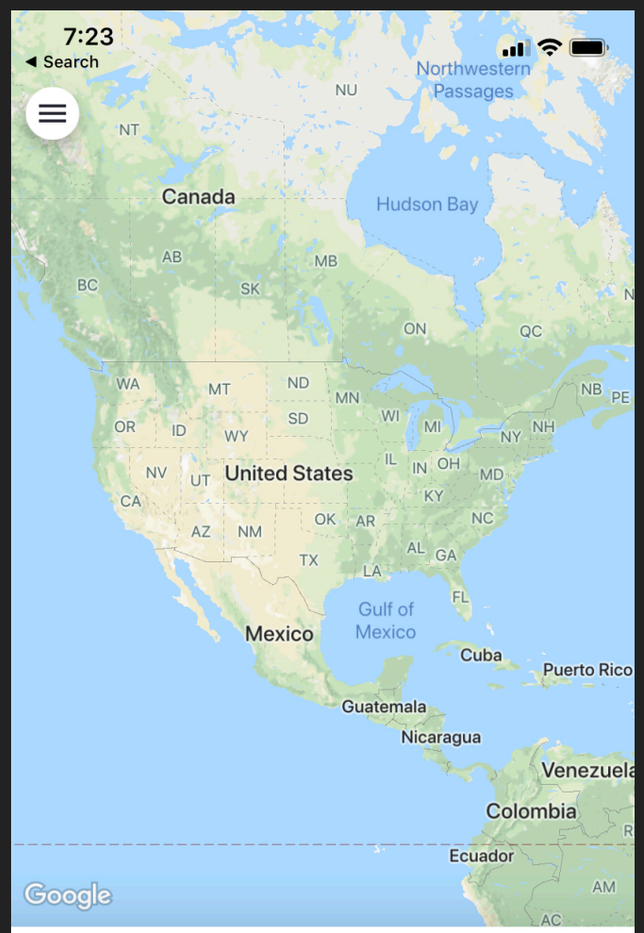
https://www.natashatherobot.com/ios-taking-the-user-to-settings/

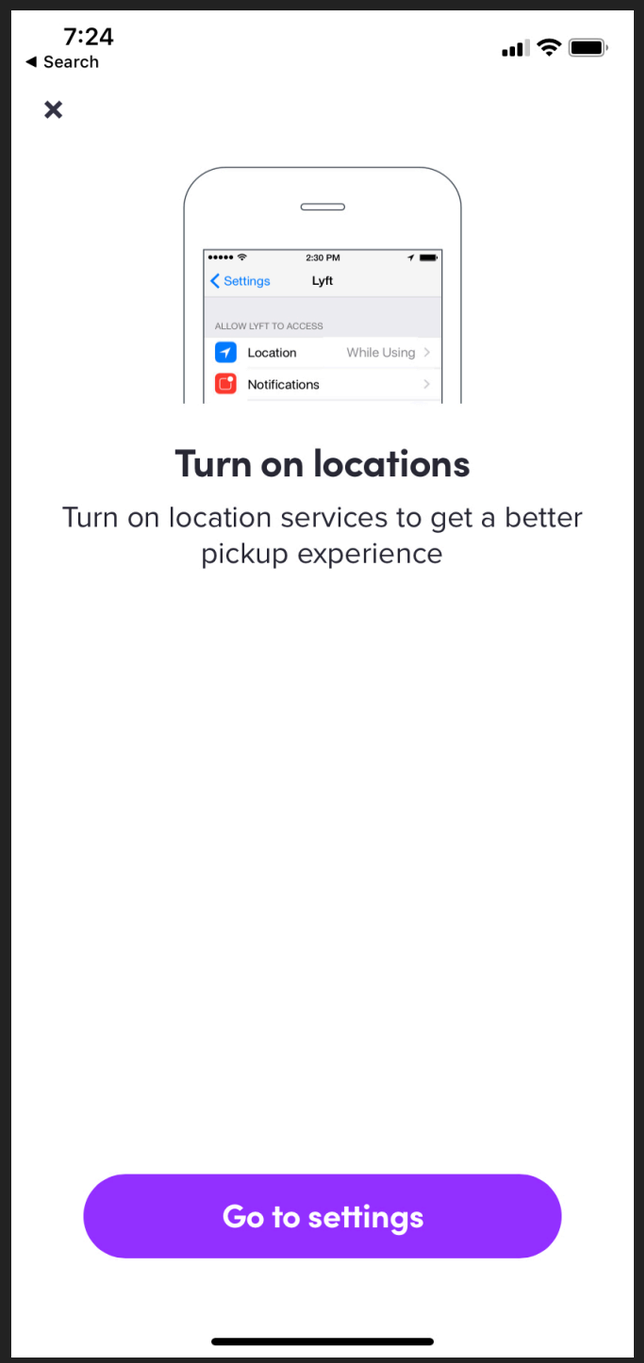
NEW METHOD

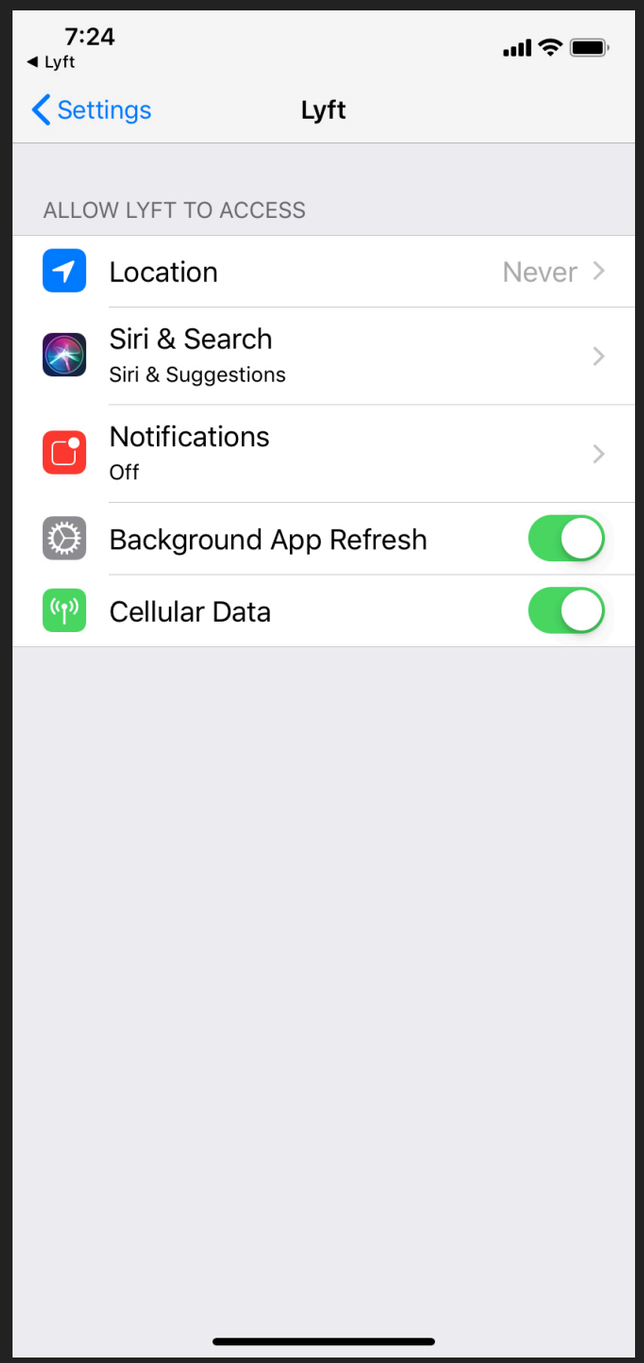
- Deep link the user to your app's specific page in Settings

```
if let settingsUrl = URL(string: UIApplication.openSettingsURLString) {
    UIApplication.shared.open(settingsUrl)
}
```

EXAMPLE: LYFT





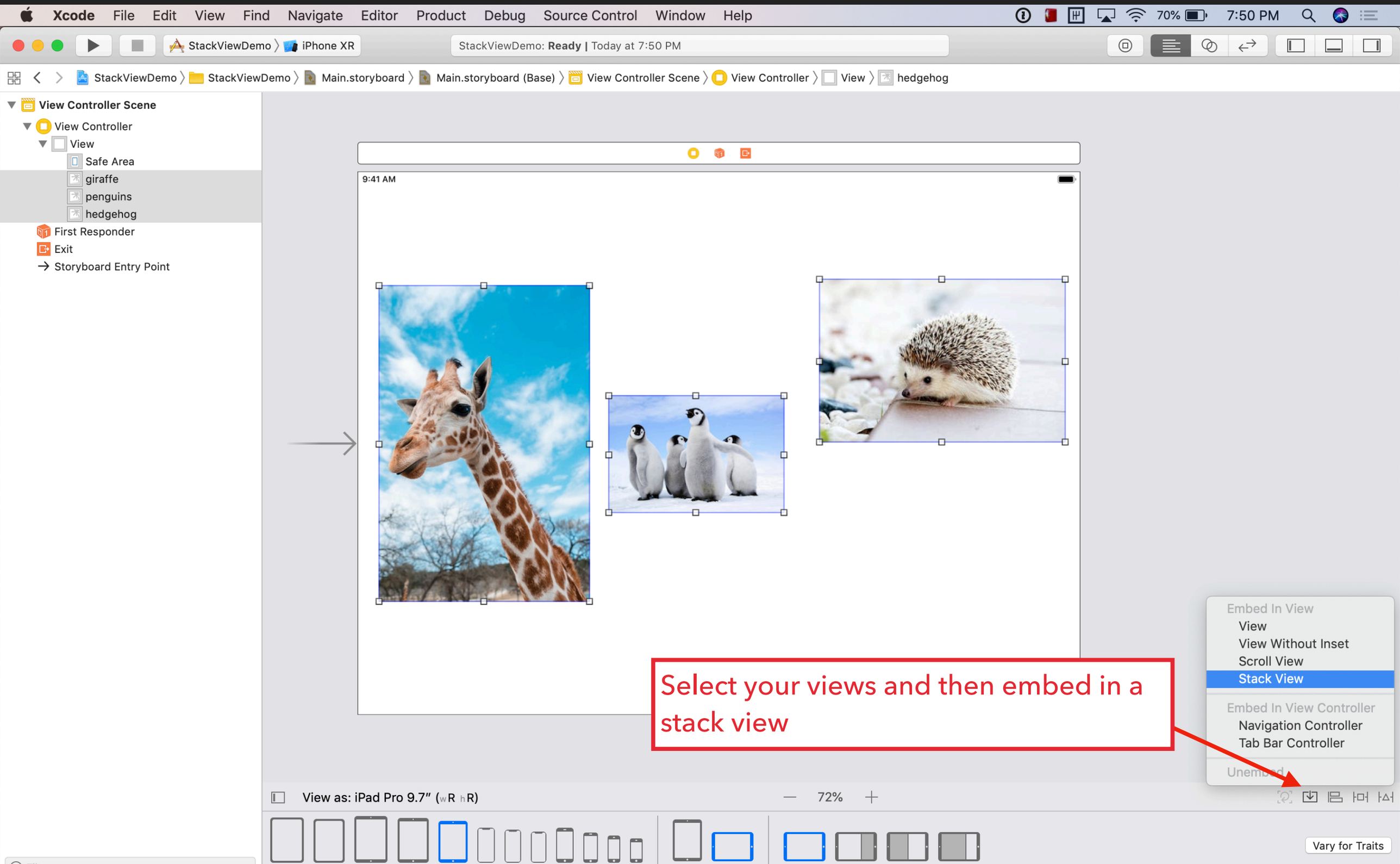


STACK VIEWS

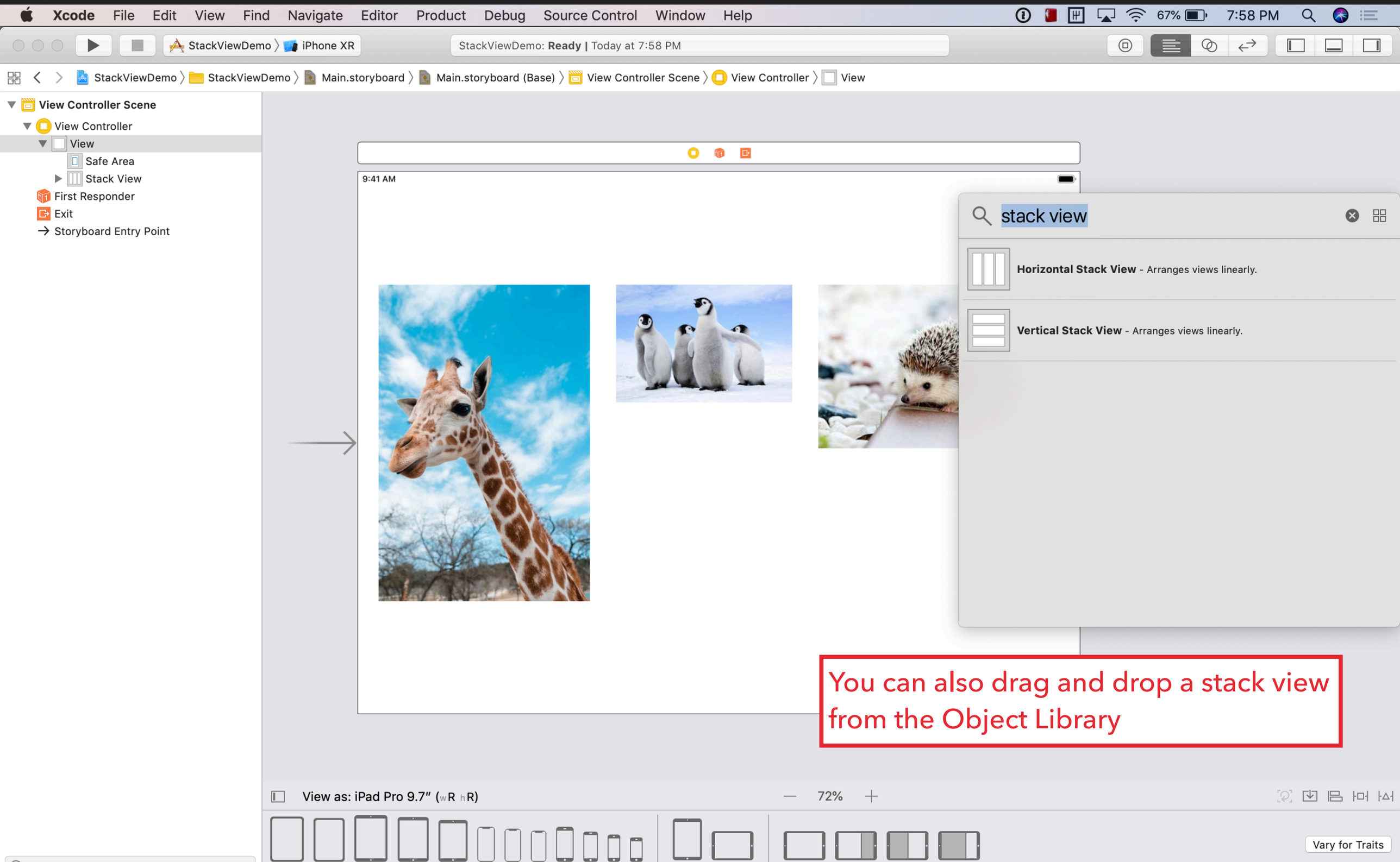
WHAT ARE STACK VIEWS?

- Containers for laying out views
- Auto Layout feature introduced in iOS 9
- Stack views are not rendered on screen (only their subviews are)

STACK VIEWS


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
70% 7:50 PM
StackViewDemo 〉 iPhone XR
StackViewDemo: Ready | Today at 7:50 PM
StackViewDemo 〉 StackViewDemo 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View 〉 hedgehog
View Controller Scene
View Controller
View
Safe Area
giraffe
penguins
hedgehog
First Responder
Exit
Storyboard Entry Point
9:41 AM
Select your views and then embed in a stack view
Embed In View
View
View Without Inset
Scroll View
Stack View
Embed In View Controller
Navigation Controller
Tab Bar Controller
View as: iPad Pro 9.7" (wR hR)
72%
Device
Orientation
Adaptation
Vary for Traits
Filter

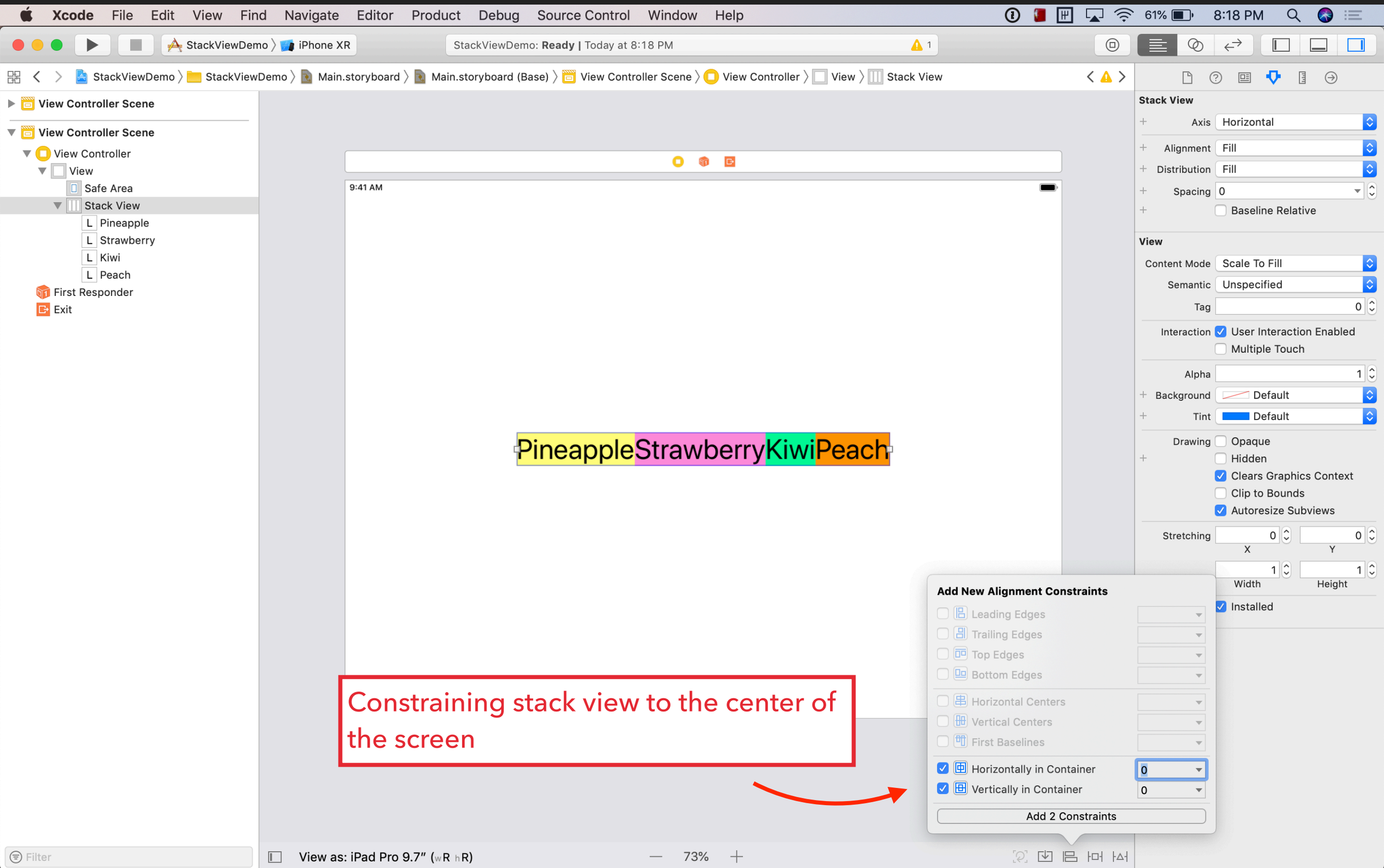
STACK VIEWS


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
67% 7:58 PM
StackViewDemo 〉 iPhone XR
StackViewDemo: Ready | Today at 7:58 PM
StackViewDemo 〉 StackViewDemo 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View
View Controller Scene
View Controller
View
Safe Area
Stack View
First Responder
Exit
Storyboard Entry Point
9:41 AM
stack view
Horizontal Stack View - Arranges views linearly.
Vertical Stack View - Arranges views linearly.
You can also drag and drop a stack view from the Object Library
View as: iPad Pro 9.7" (wR hR)
72%
Device
Orientation
Adaptation
Vary for Traits
Filter

POSITIONING YOUR STACK VIEW

- Use auto layout constraints to define the position and (optionally) size of your stack view
    - If you do not define a size, the contents of the stack view will determine its height and width

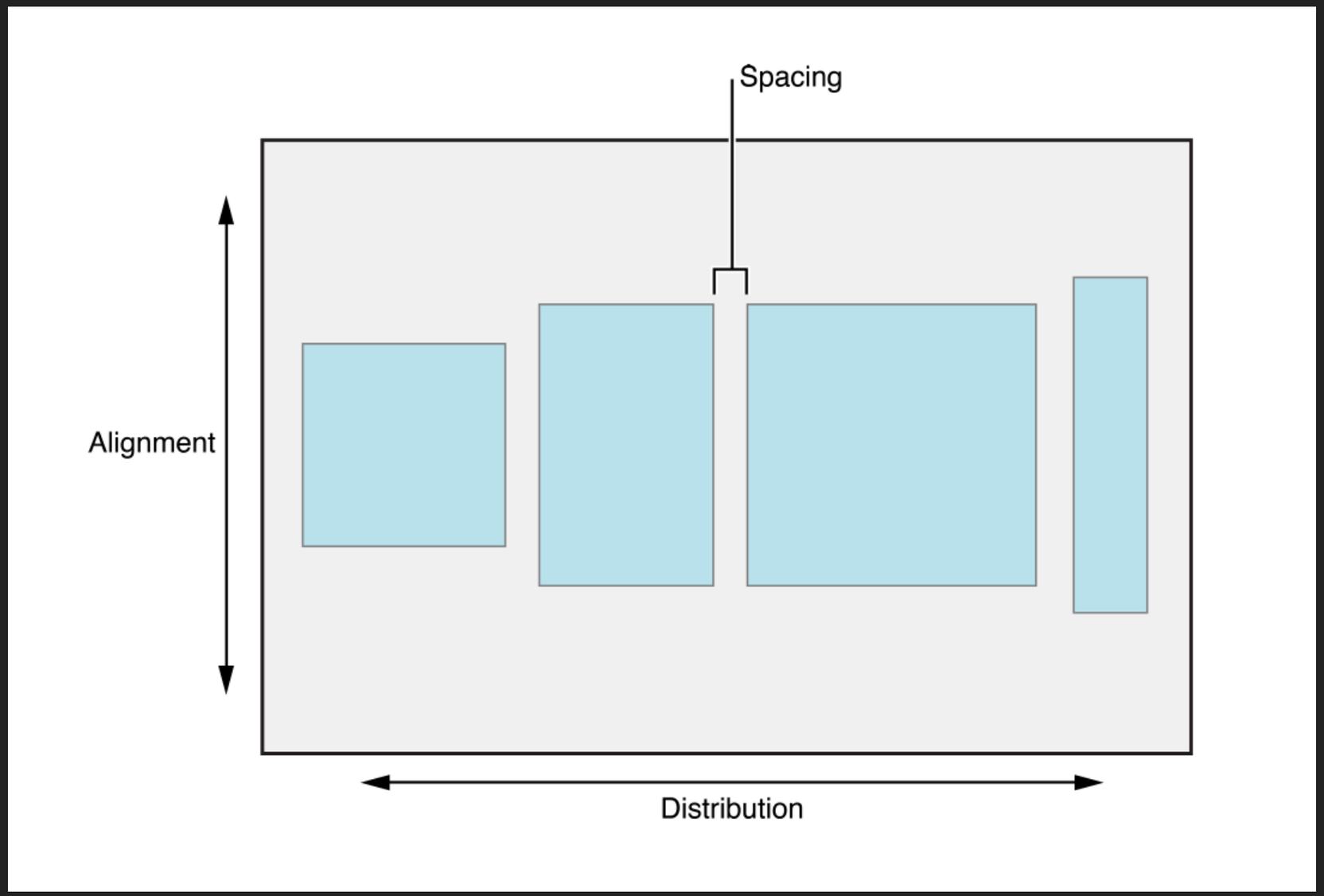
STACK VIEWS


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
61% 8:18 PM
StackViewDemo iPhone XR
StackViewDemo: Ready | Today at 8:18 PM
StackViewDemo StackViewDemo Main.storyboard Main.storyboard (Base) View Controller Scene View Controller View Stack View
View Controller Scene
View Controller Scene
View Controller
View
Safe Area
Stack View
Pineapple
Strawberry
Kiwi
Peach
First Responder
Exit
9:41 AM
PineappleStrawberryKiwiPeach
Stack View
Axis Horizontal
Alignment Fill
Distribution Fill
Spacing 0
Baseline Relative
View
Content Mode Scale To Fill
Semantic Unspecified
Tag 0
Interaction User Interaction Enabled
Multiple Touch
Alpha 1
Background Default
Tint Default
Drawing Opaque
Hidden
Clears Graphics Context
Clip to Bounds
Autoresize Subviews
Stretching 0 X 0 Y
1 Width 1 Height
Installed
Add New Alignment Constraints
Leading Edges
Trailing Edges
Top Edges
Bottom Edges
Horizontal Centers
Vertical Centers
First Baselines
Horizontally in Container 0
Vertically in Container 0
Add 2 Constraints
Constraining stack view to the center of the screen
Filter
View as: iPad Pro 9.7" (wR hR)
73%

4 MAIN PROPERTIES OF A STACK VIEW

- Axis
- Distribution
- Alignment
- Spacing

4 MAIN PROPERTIES OF A STACK VIEW

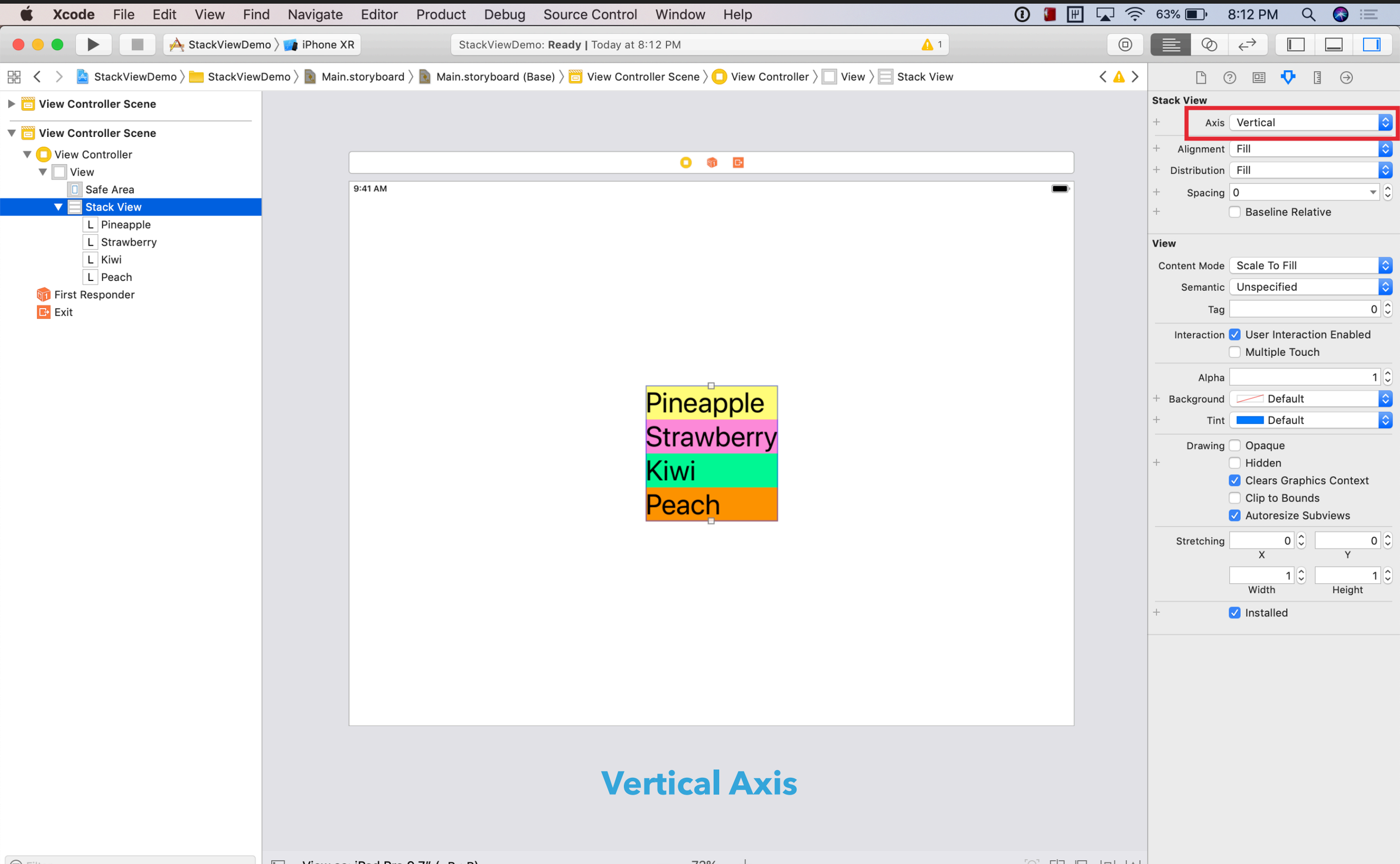


https://developer.apple.com/documentation/uikit/uistackview

AXIS

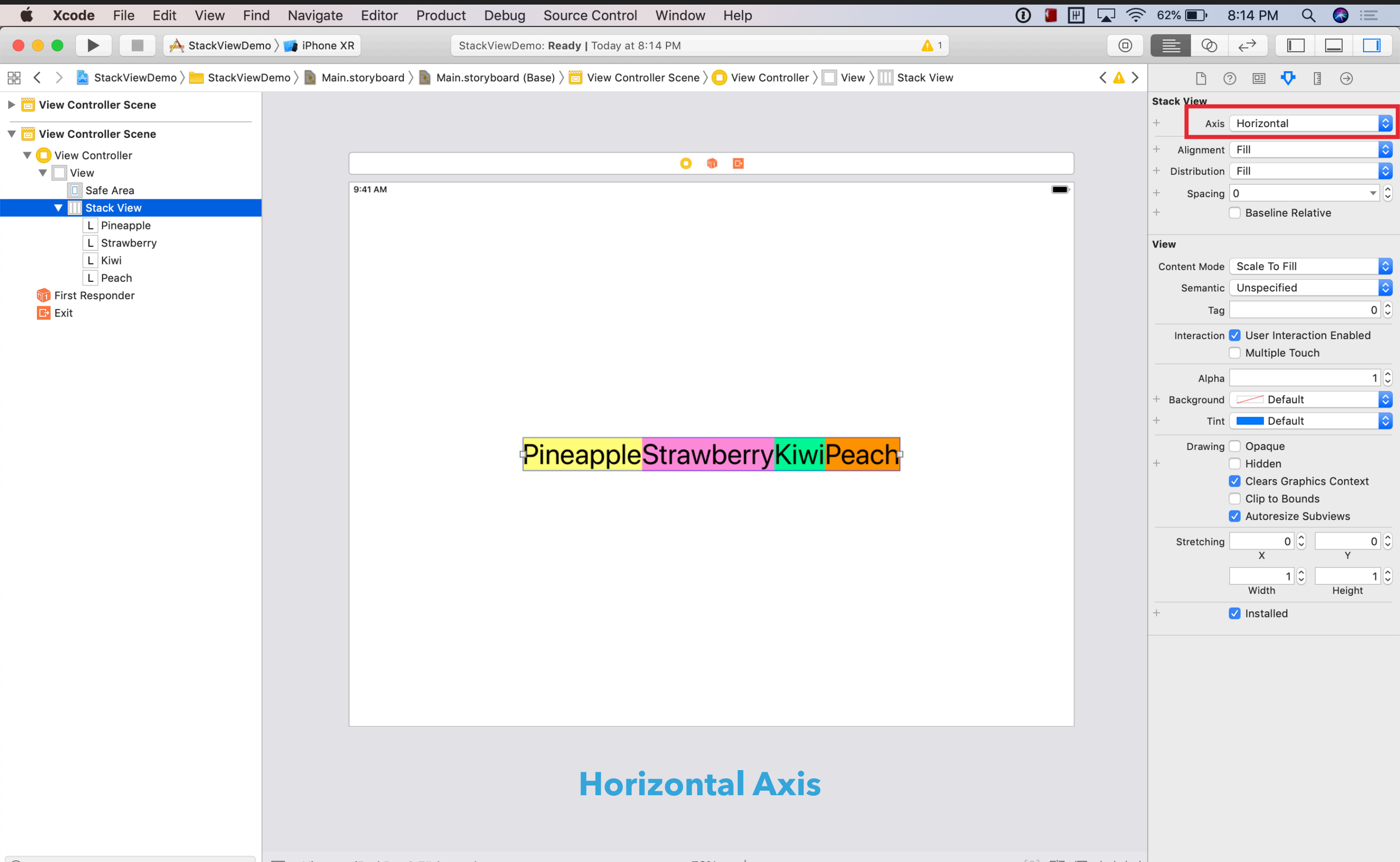
- Either vertical or horizontal

STACK VIEWS



Vertical Axis

STACK VIEWS

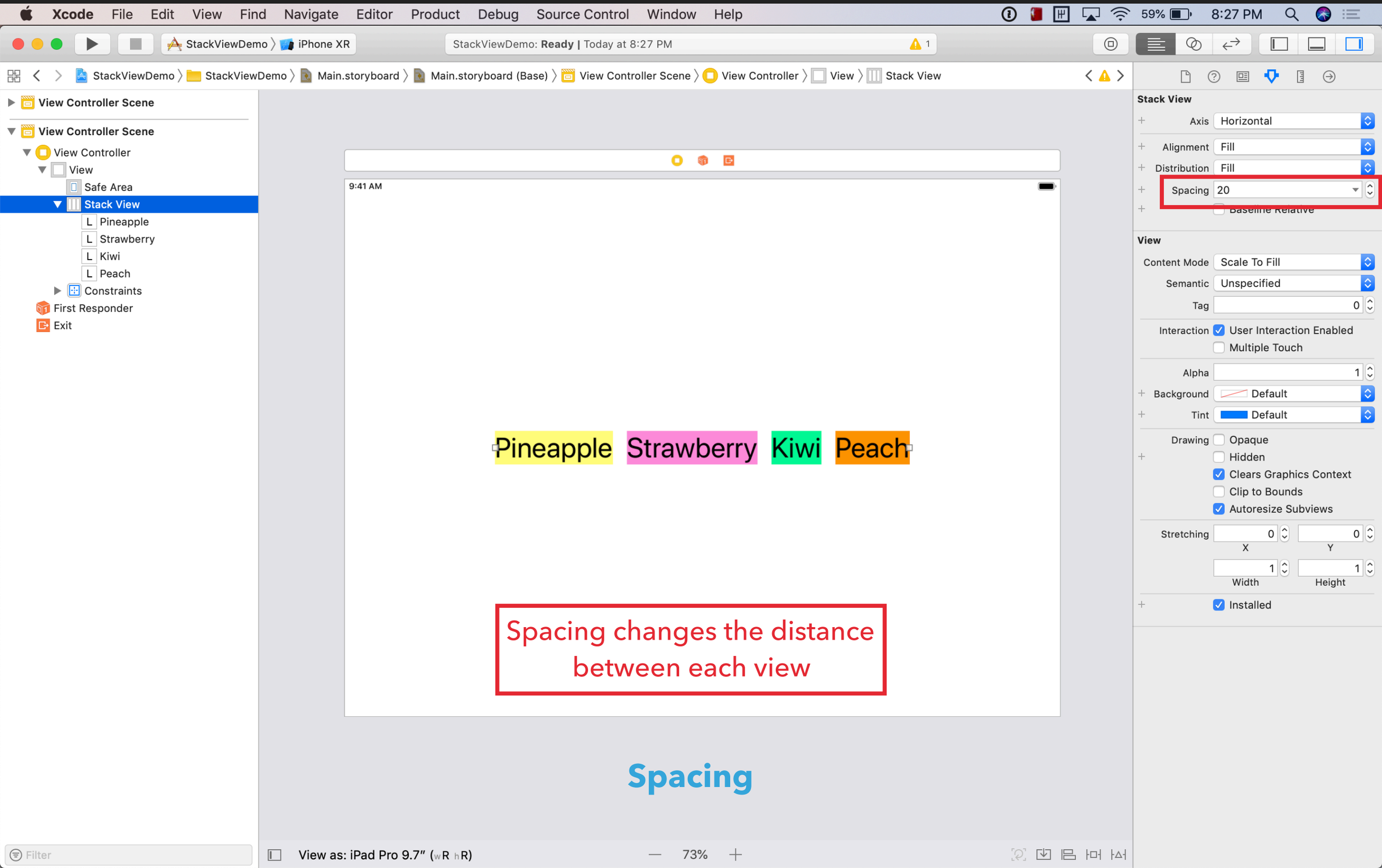


Horizontal Axis

SPACING

- Distance between views
    - May be greater than the value you specify depending on the `distribution` property
    - Can be set to a negative number to create an overlap

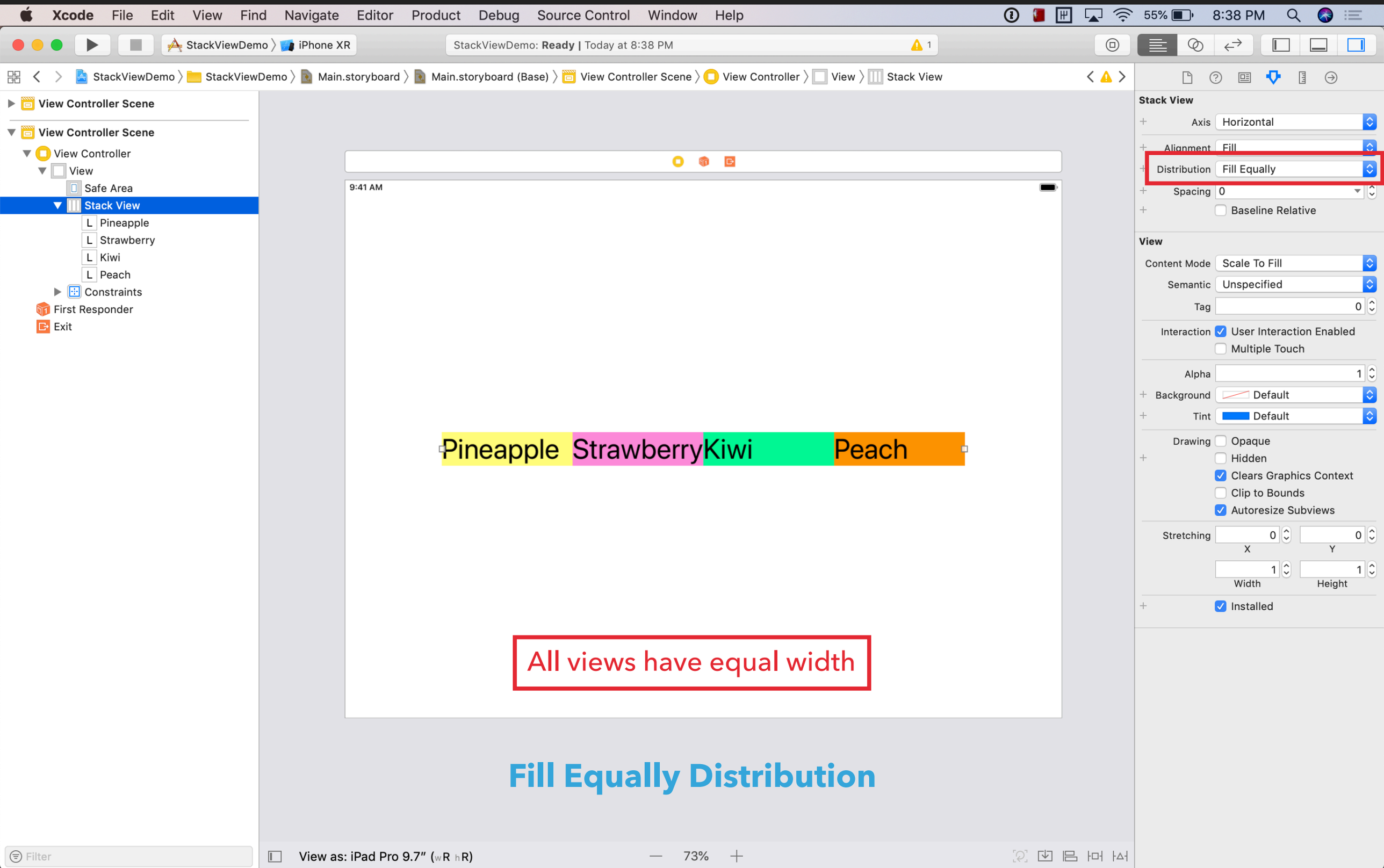
STACK VIEWS


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
59% 8:27 PM
StackViewDemo iPhone XR
StackViewDemo: Ready | Today at 8:27 PM
StackViewDemo StackViewDemo Main.storyboard Main.storyboard (Base) View Controller Scene View Controller View Stack View
View Controller Scene
View Controller Scene
View Controller
View
Safe Area
Stack View
Pineapple
Strawberry
Kiwi
Peach
Constraints
First Responder
Exit
9:41 AM
Pineapple Strawberry Kiwi Peach
Spacing changes the distance between each view
Spacing
Stack View
Axis Horizontal
Alignment Fill
Distribution Fill
Spacing 20
Baseline Relative
View
Content Mode Scale To Fill
Semantic Unspecified
Tag 0
Interaction User Interaction Enabled
Multiple Touch
Alpha 1
Background Default
Tint Default
Drawing Opaque
Hidden
Clears Graphics Context
Clip to Bounds
Autoresize Subviews
Stretching 0 X 0 Y
1 Width 1 Height
Installed
Filter
View as: iPad Pro 9.7" (wR hR)
73%

DISTRIBUTION

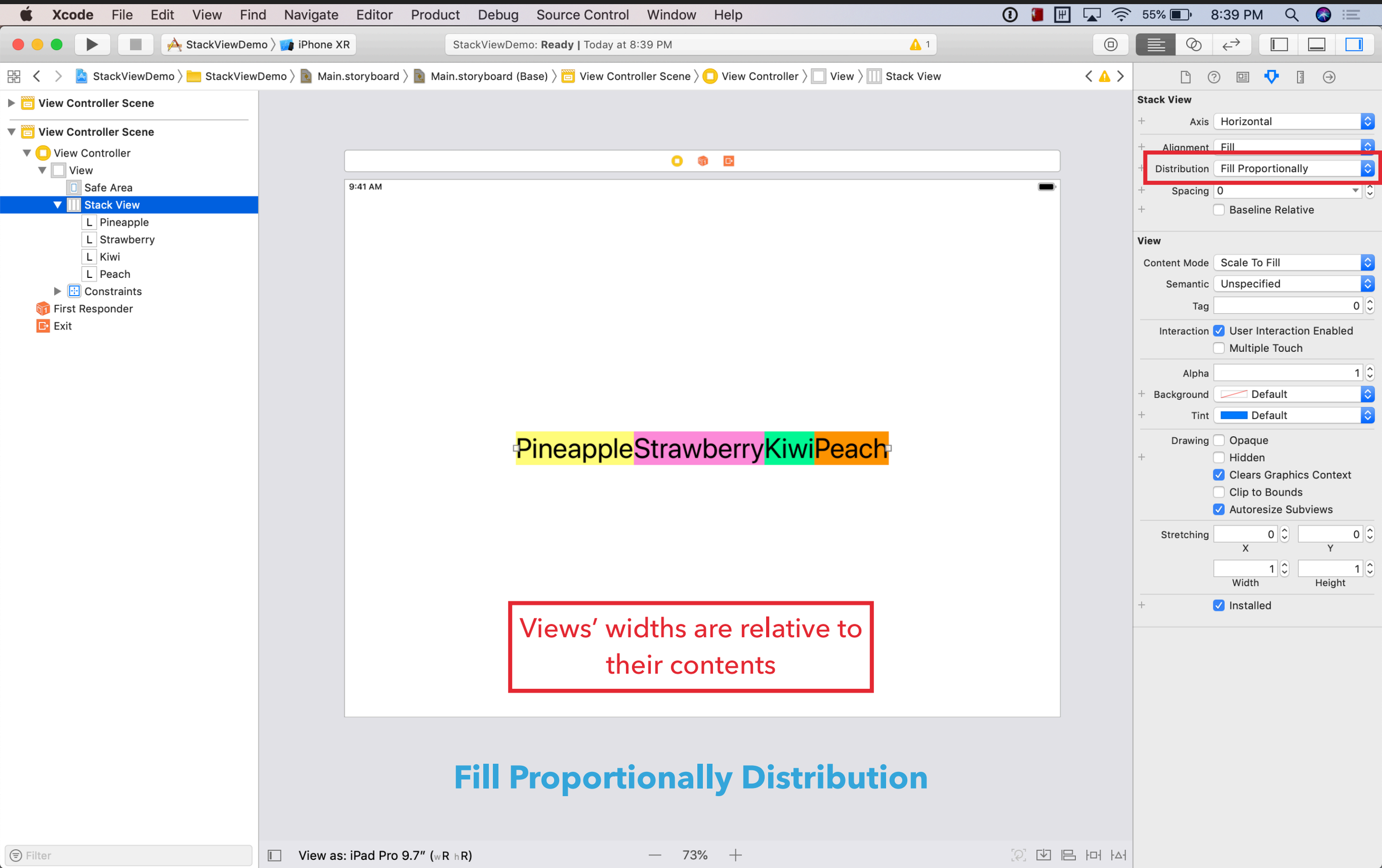
- Defines the size and position of views along the stack view's axis

STACK VIEWS



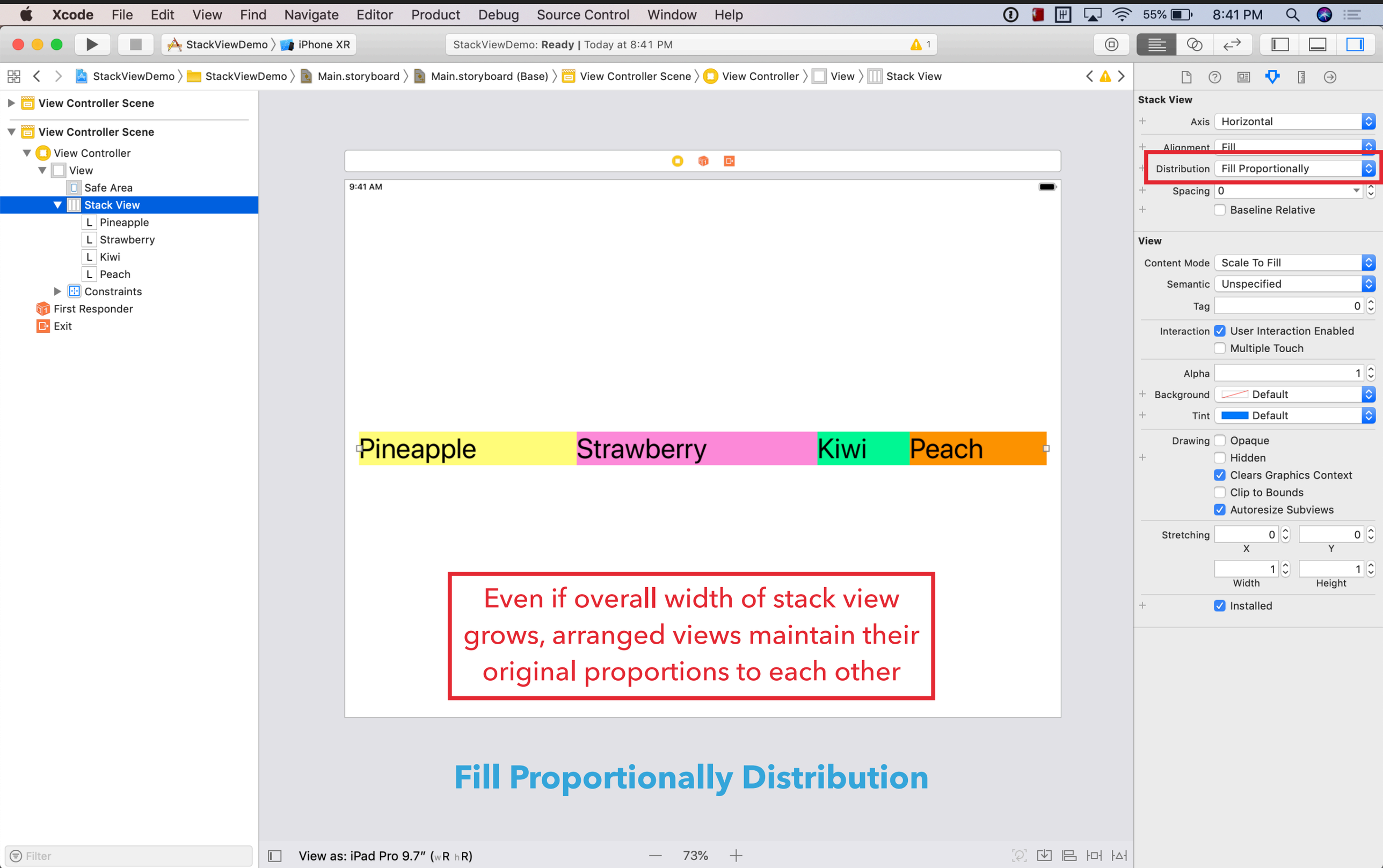
Fill Equally Distribution

STACK VIEWS

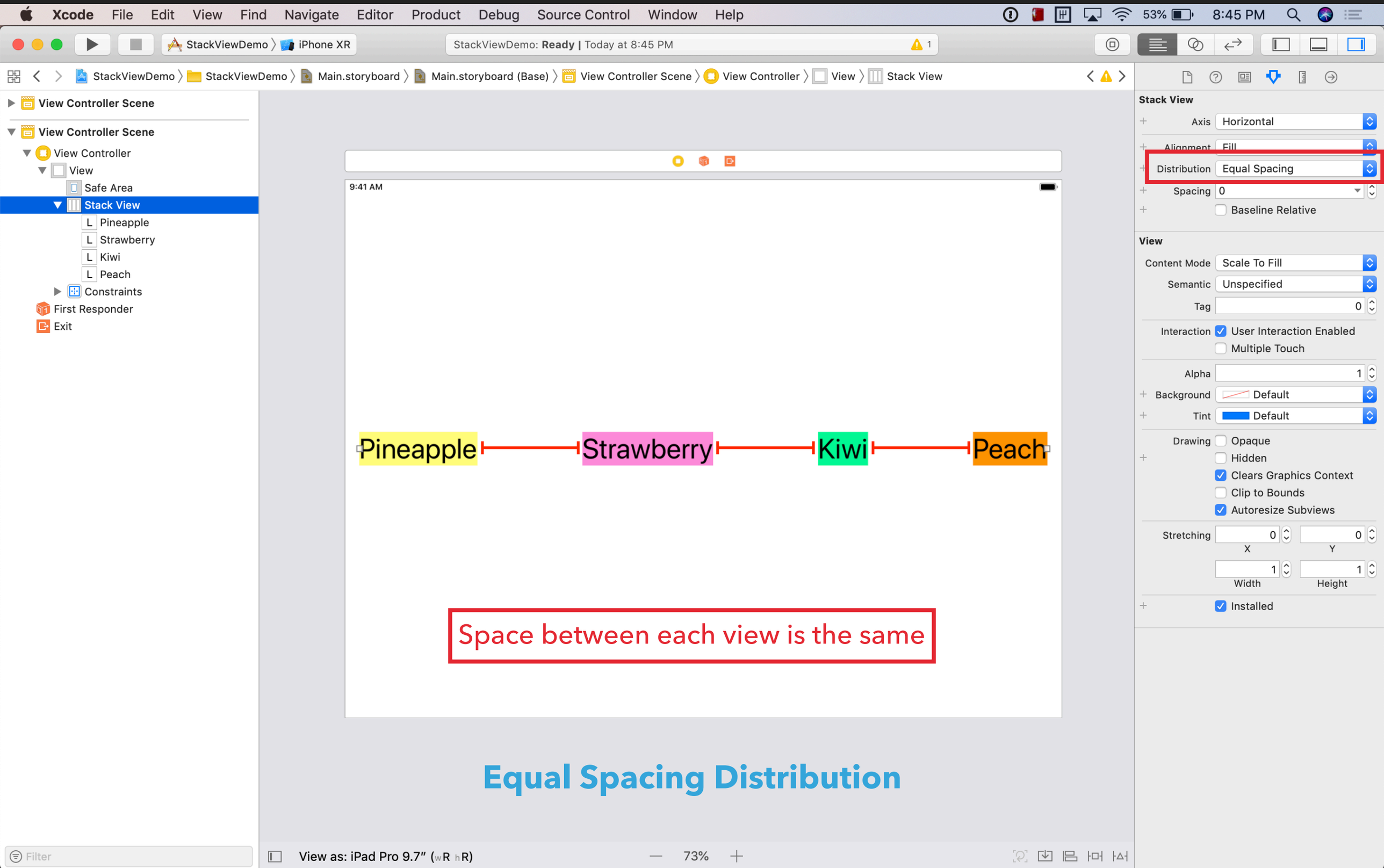


Fill Proportionally Distribution

STACK VIEWS

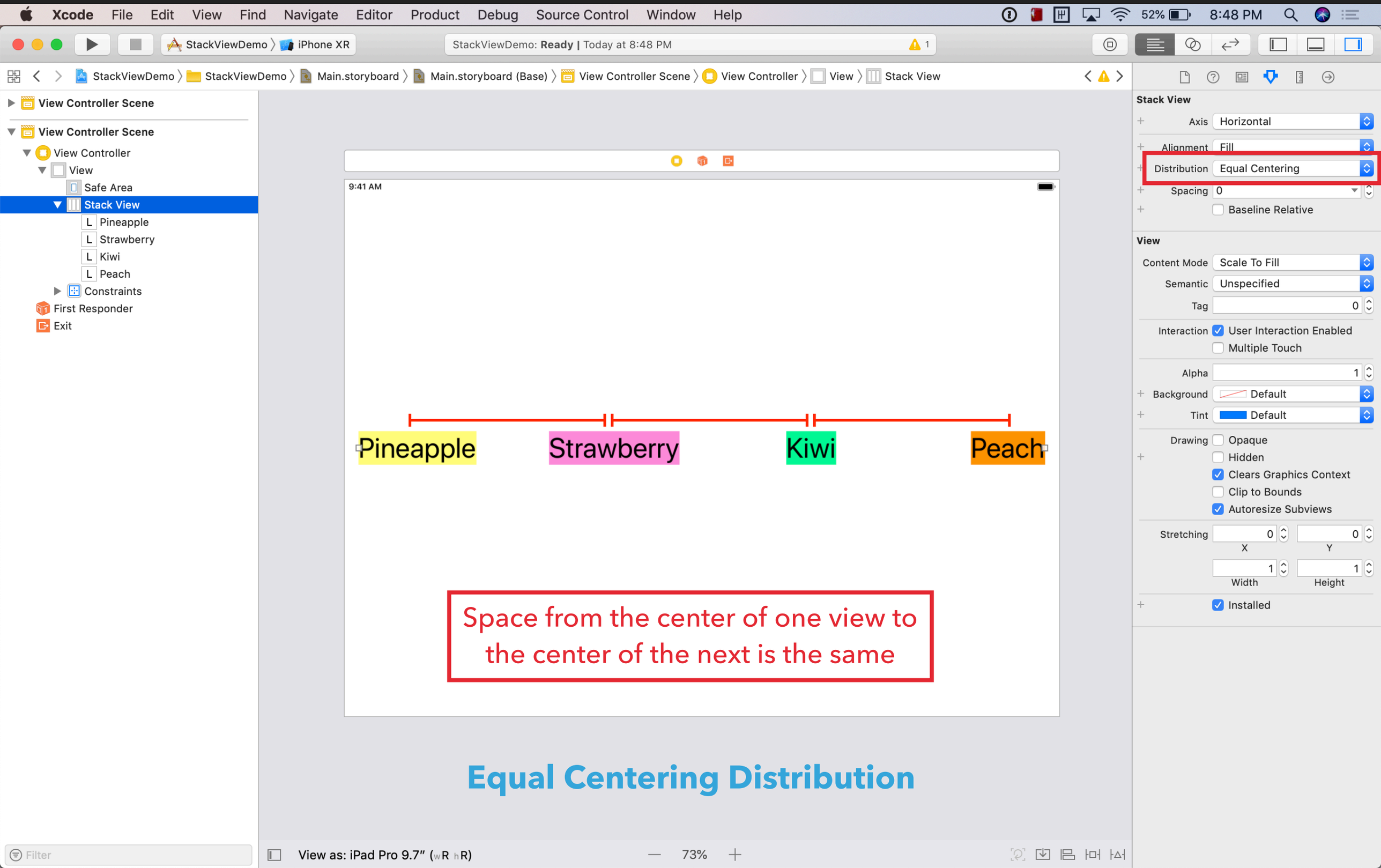

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
55% 8:41 PM
StackViewDemo iPhone XR
StackViewDemo: Ready | Today at 8:41 PM
StackViewDemo StackViewDemo Main.storyboard Main.storyboard (Base) View Controller Scene View Controller View Stack View
View Controller Scene
View Controller Scene
View Controller
View
Safe Area
Stack View
Pineapple
Strawberry
Kiwi
Peach
Constraints
First Responder
Exit
9:41 AM
Pineapple
Strawberry
Kiwi
Peach
Even if overall width of stack view grows, arranged views maintain their original proportions to each other
Fill Proportionally Distribution
Stack View
Axis Horizontal
Alignment Fill
Distribution Fill Proportionally
Spacing 0
Baseline Relative
View
Content Mode Scale To Fill
Semantic Unspecified
Tag 0
Interaction User Interaction Enabled
Multiple Touch
Alpha 1
Background Default
Tint Default
Drawing Opaque
Hidden
Clears Graphics Context
Clip to Bounds
Autoresize Subviews
Stretching 0 X 0 Y
1 Width 1 Height
Installed
Filter
View as: iPad Pro 9.7" (wR hR)
73%

STACK VIEWS



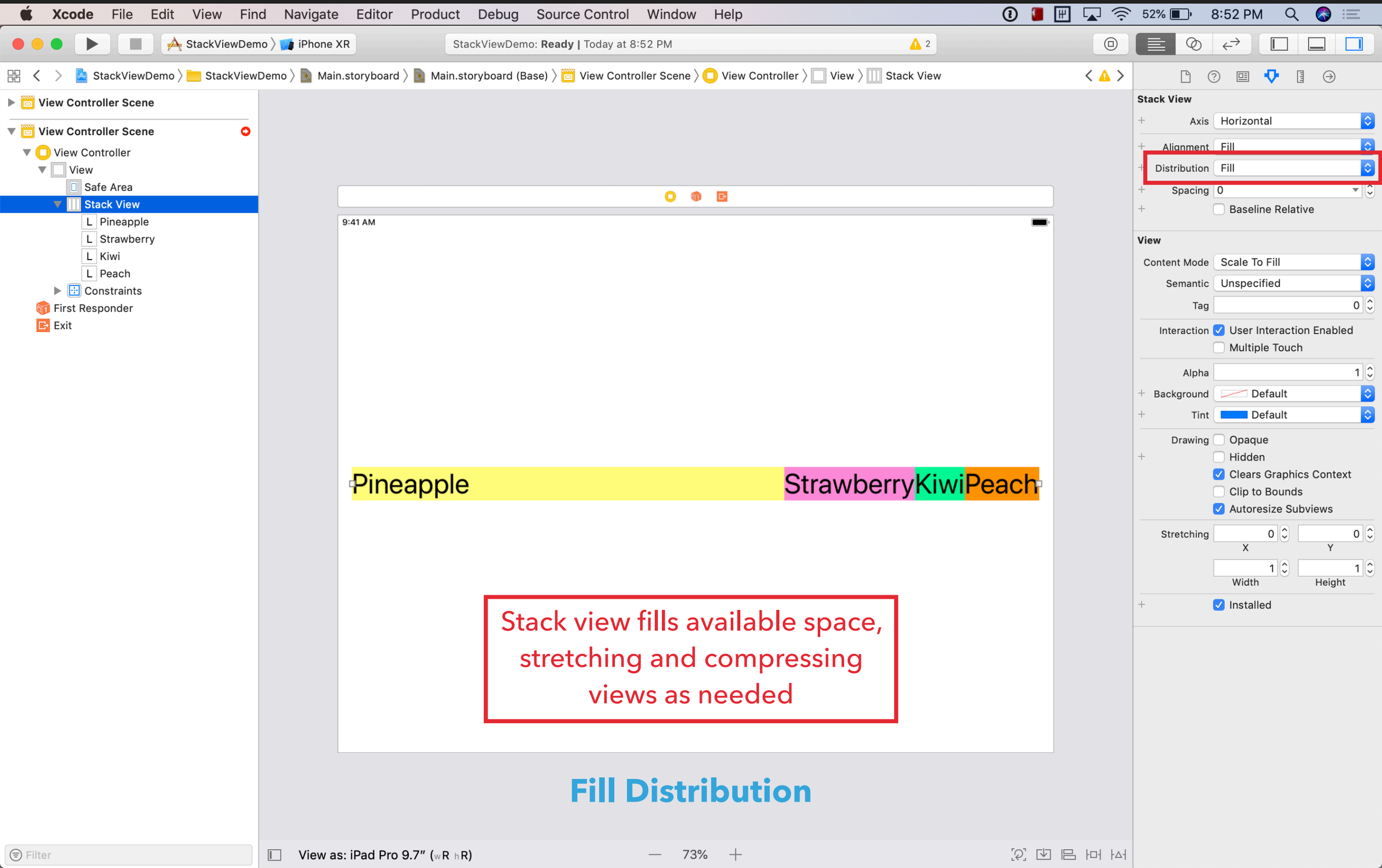
Equal Spacing Distribution

STACK VIEWS

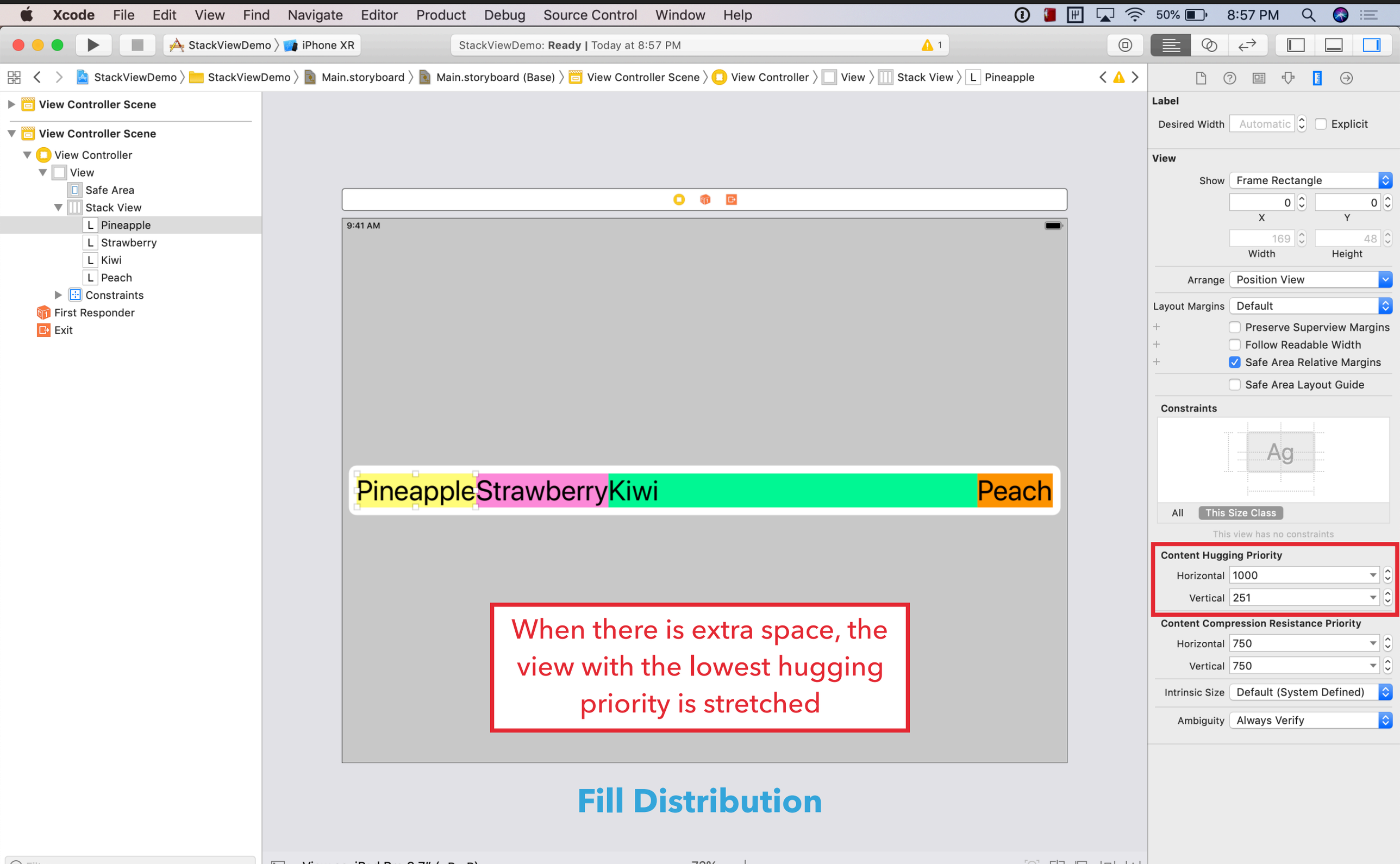


Equal Centering Distribution

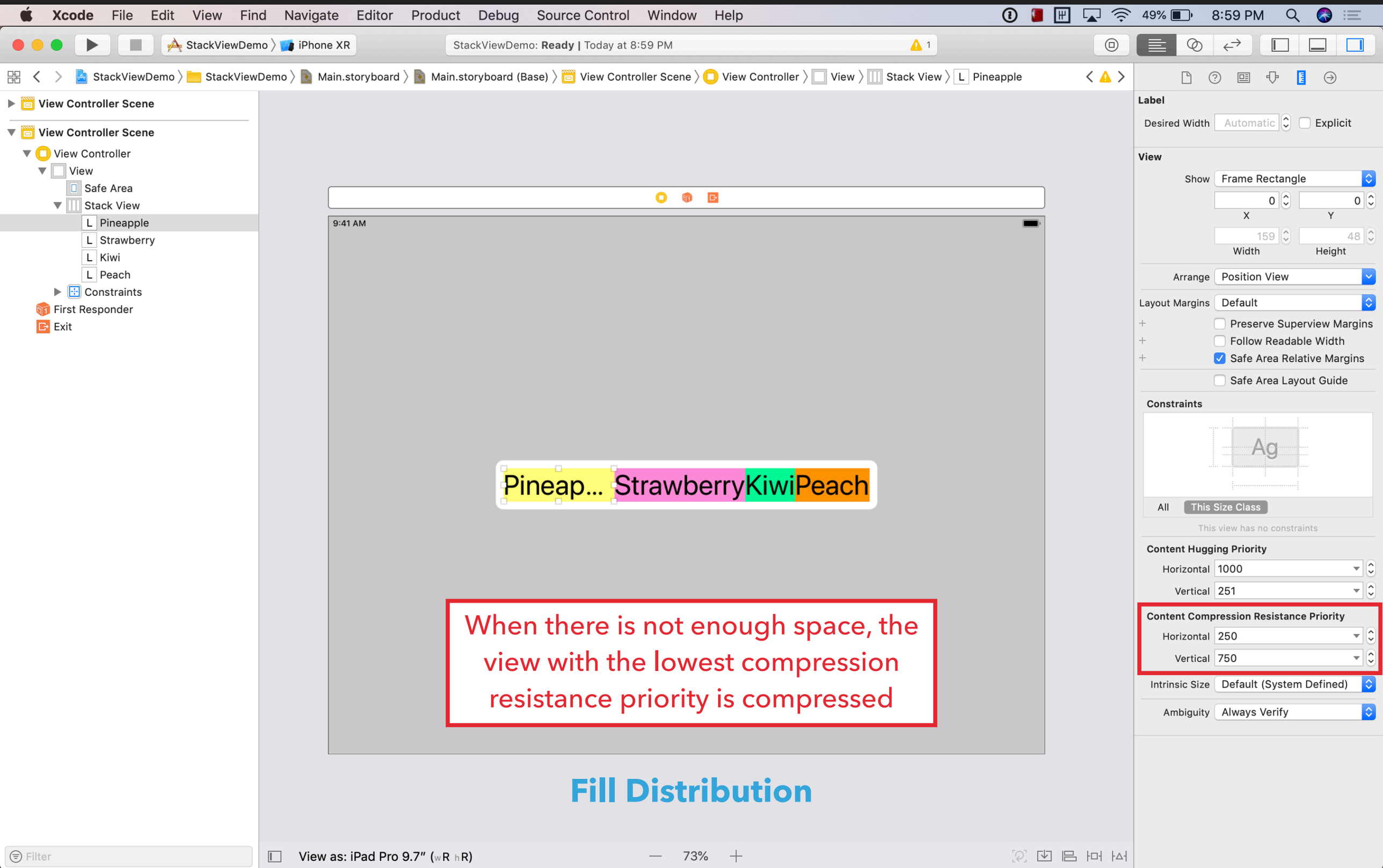
STACK VIEWS


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
52% 8:52 PM
StackViewDemo 〉 iPhone XR
StackViewDemo: Ready | Today at 8:52 PM
StackViewDemo 〉 StackViewDemo 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View 〉 Stack View
View Controller Scene
View Controller Scene
View Controller
View
Safe Area
Stack View
Pineapple
Strawberry
Kiwi
Peach
Constraints
First Responder
Exit
9:41 AM
Pineapple
StrawberryKiwiPeach
Stack view fills available space, stretching and compressing views as needed
Fill Distribution
Stack View
Axis Horizontal
Alignment Fill
Distribution Fill
Spacing 0
Baseline Relative
View
Content Mode Scale To Fill
Semantic Unspecified
Tag 0
Interaction User Interaction Enabled
Multiple Touch
Alpha 1
Background Default
Tint Default
Drawing Opaque
Hidden
Clears Graphics Context
Clip to Bounds
Autoresize Subviews
Stretching 0 X 0 Y
1 Width 1 Height
Installed
Filter
View as: iPad Pro 9.7" (wR hR)
73%

STACK VIEWS


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
50% 8:57 PM
StackViewDemo iPhone XR
StackViewDemo: Ready | Today at 8:57 PM
StackViewDemo StackViewDemo Main.storyboard Main.storyboard (Base) View Controller Scene View Controller View Stack View Pineapple
View Controller Scene
View Controller Scene
View Controller
View
Safe Area
Stack View
Pineapple
Strawberry
Kiwi
Peach
Constraints
First Responder
Exit
9:41 AM
PineappleStrawberryKiwiPeach
When there is extra space, the view with the lowest hugging priority is stretched
Fill Distribution
Label
Desired Width Automatic Explicit
View
Show Frame Rectangle
0 0
X Y
169 48
Width Height
Arrange Position View
Layout Margins Default
Preserve Superview Margins
Follow Readable Width
Safe Area Relative Margins
Safe Area Layout Guide
Constraints
Ag
All This Size Class
This view has no constraints
Content Hugging Priority
Horizontal 1000
Vertical 251
Content Compression Resistance Priority
Horizontal 750
Vertical 750
Intrinsic Size Default (System Defined)
Ambiguity Always Verify
Filter
View as: iPad Pro 9.7" (wR hR)
73%

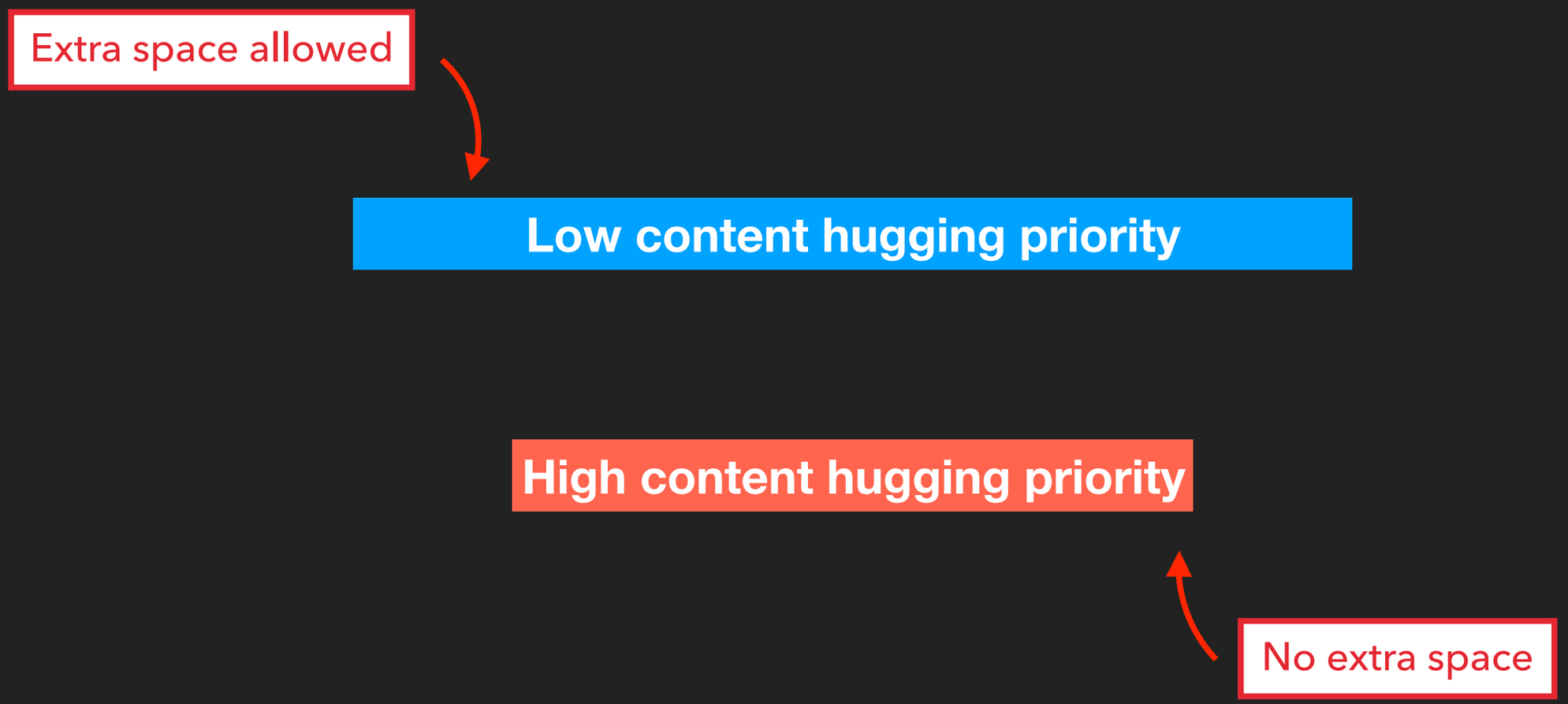
STACK VIEWS



Fill Distribution

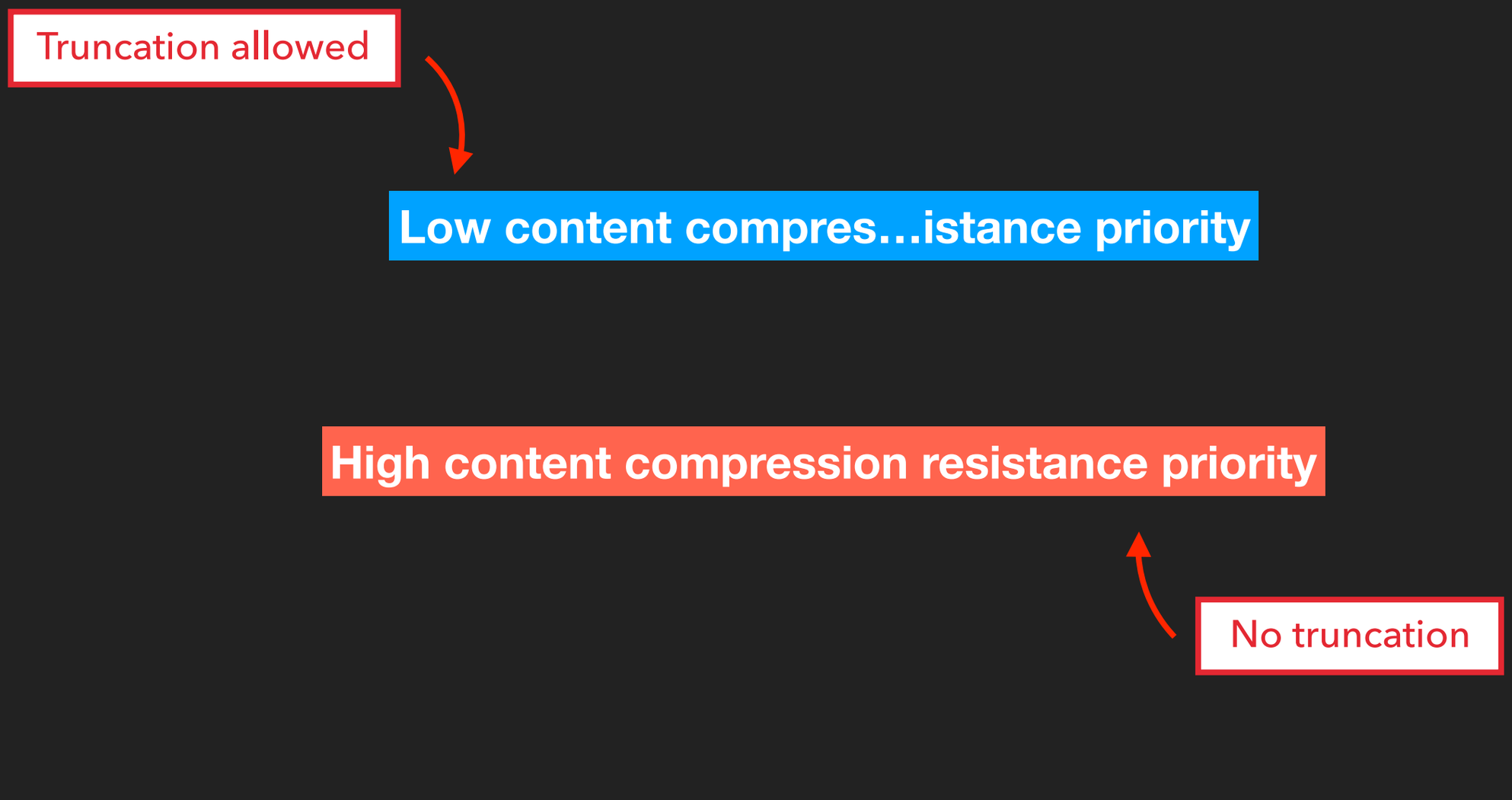
CONTENT HUGGING PRIORITY

- Set to an integer between 1 and 1000



CONTENT COMPRESSION RESISTANCE PRIORITY

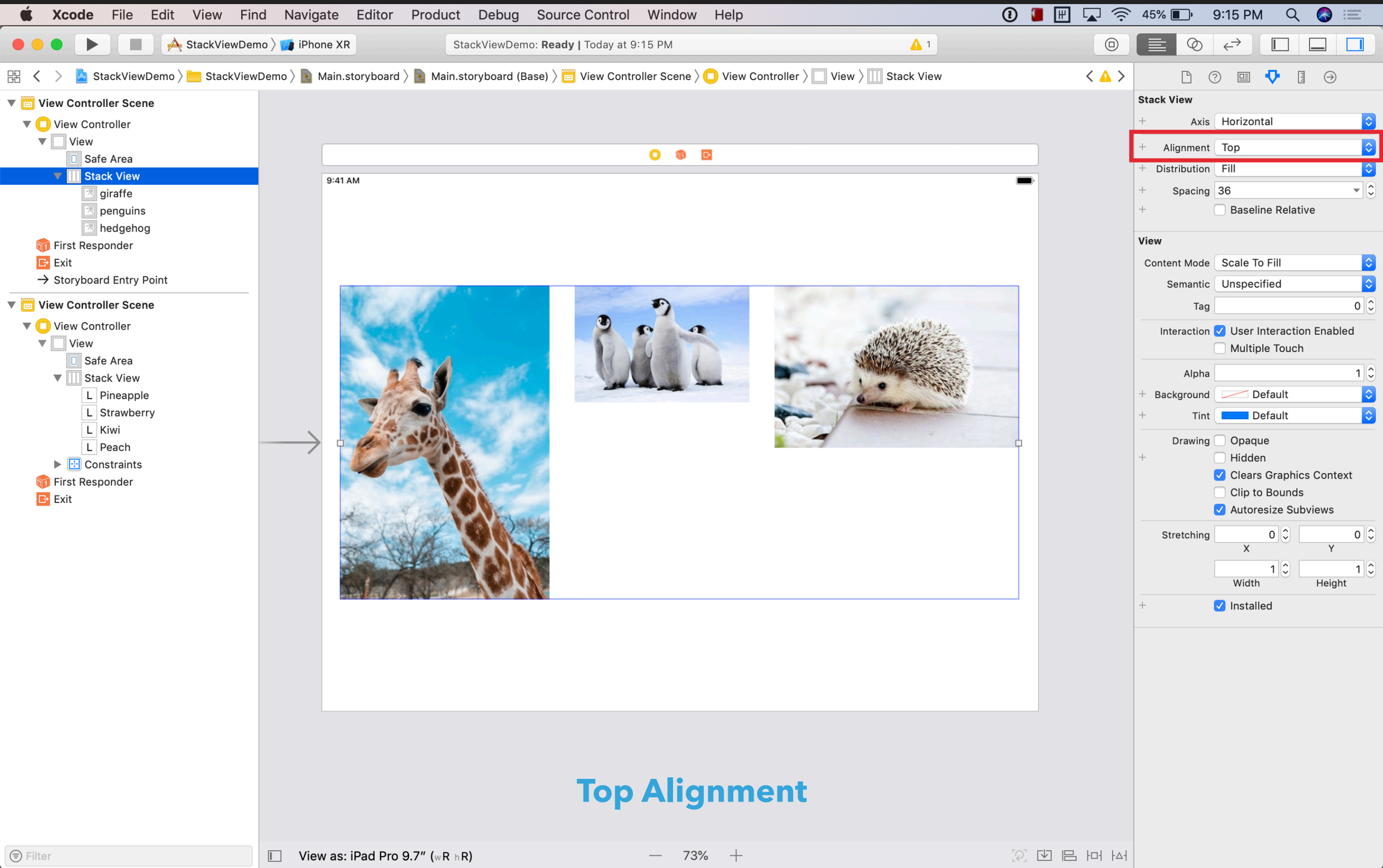
- Set to an integer between 1 and 1000



ALIGNMENT

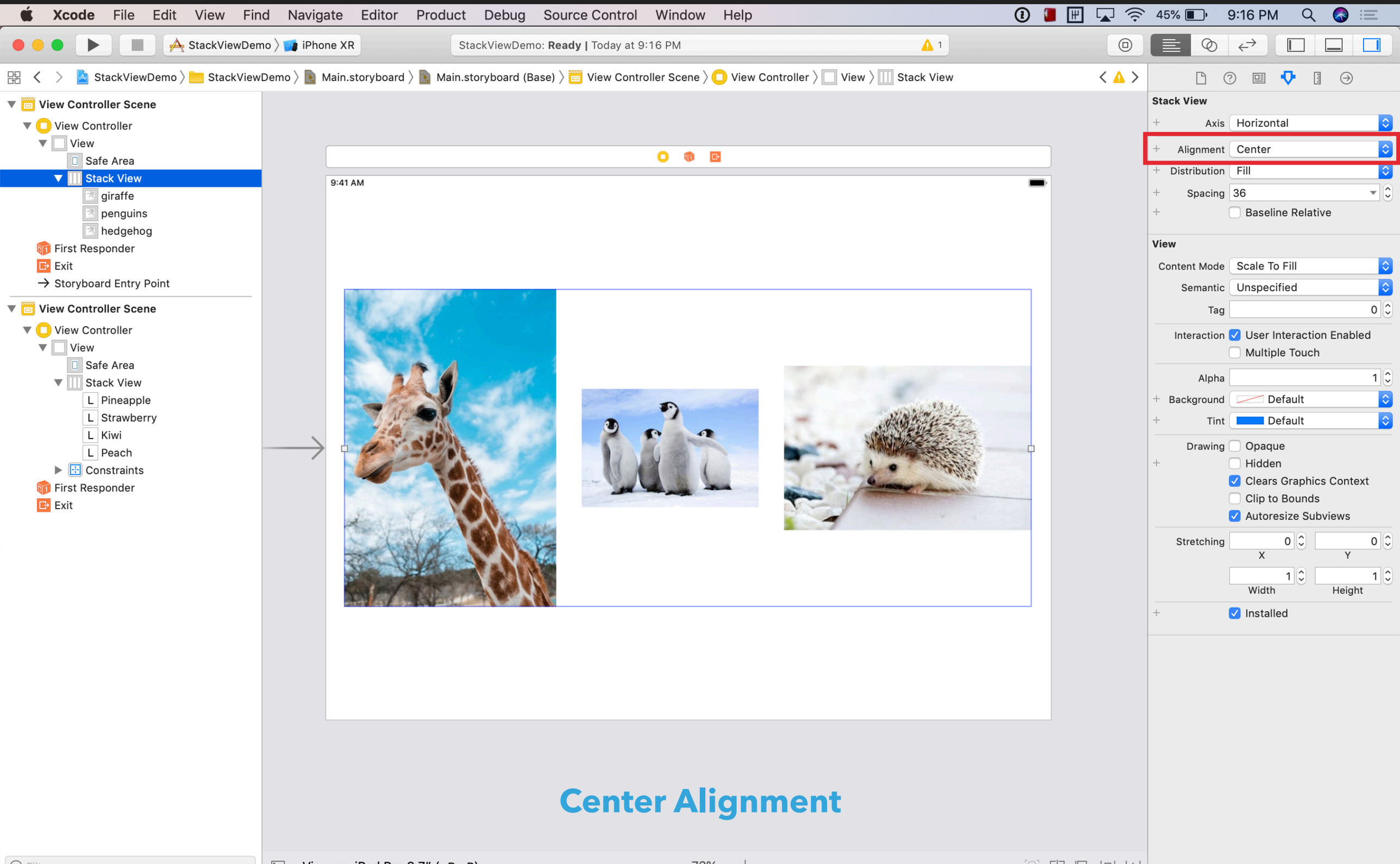
- Defines the size and position of views perpendicular to the stack view's axis

STACK VIEWS



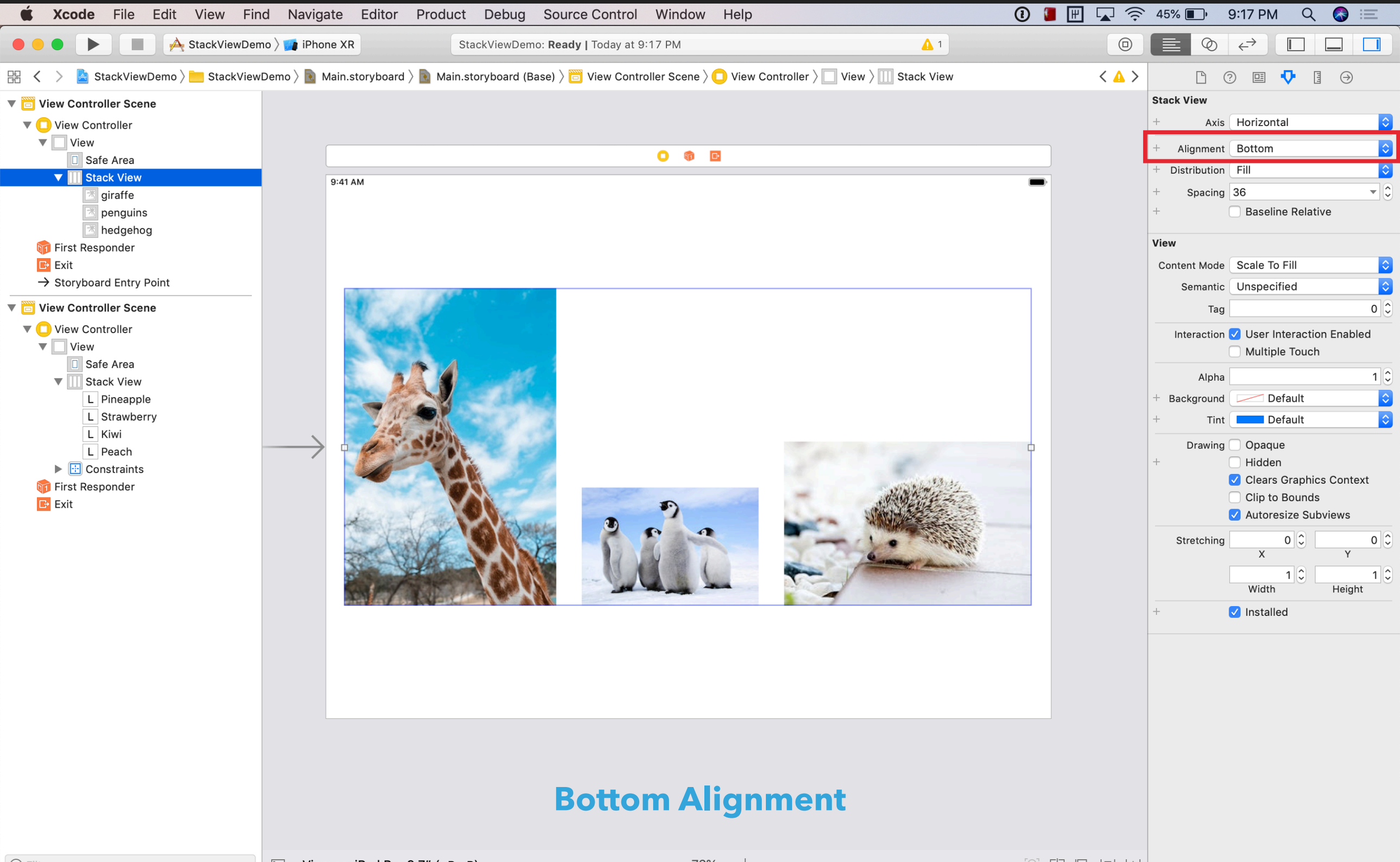
Top Alignment

STACK VIEWS



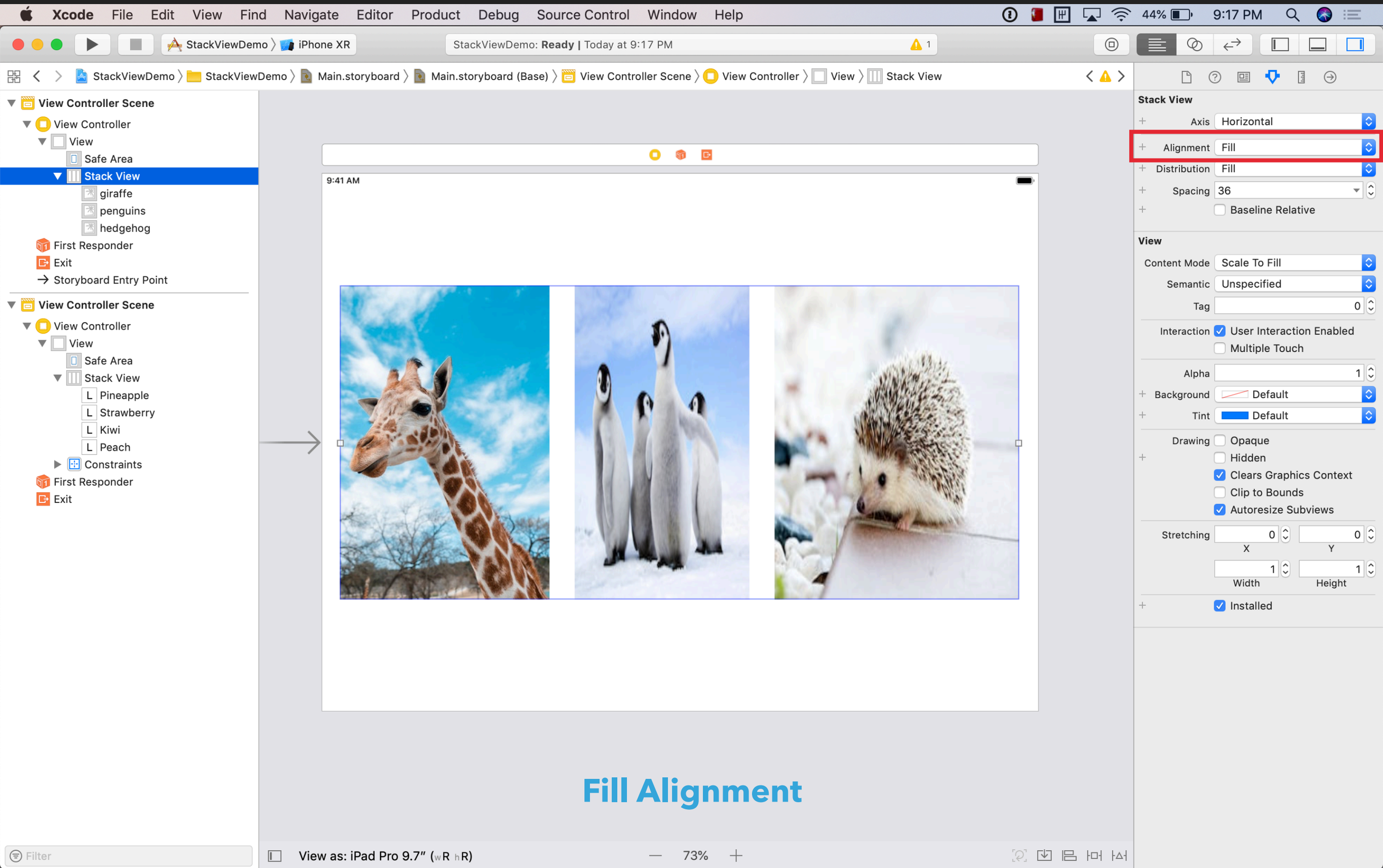
Center Alignment

STACK VIEWS



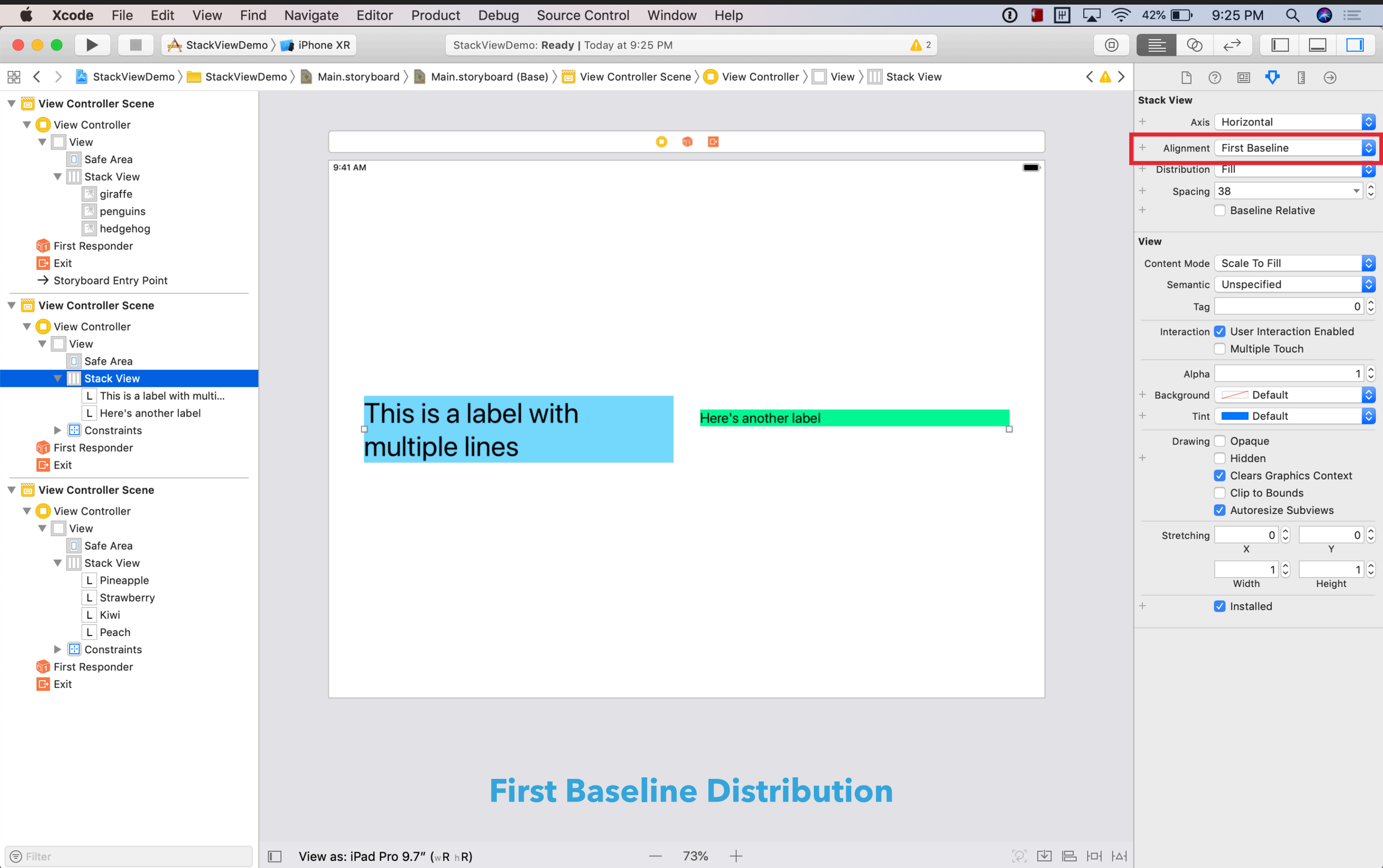
Bottom Alignment

STACK VIEWS



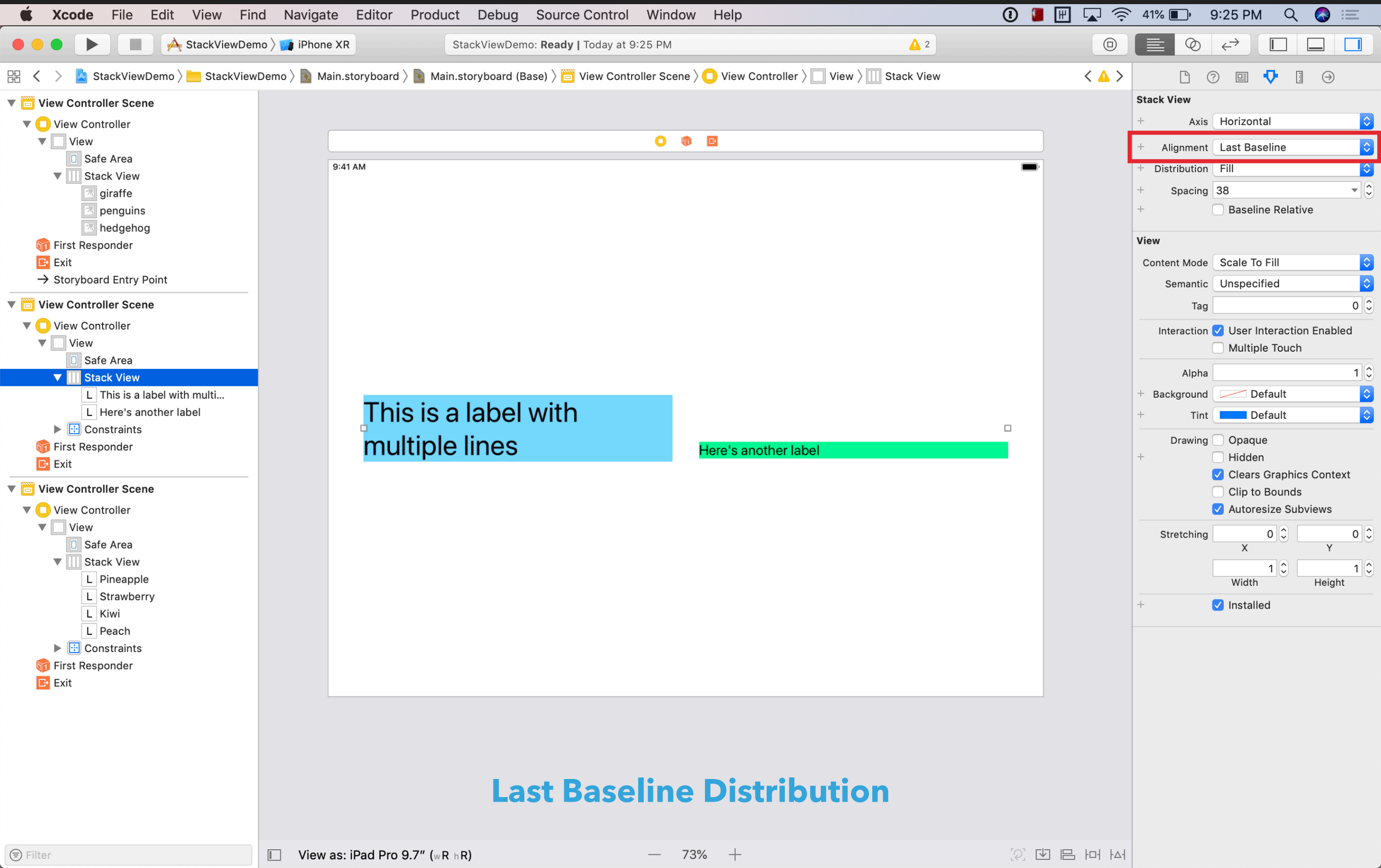
Fill Alignment

STACK VIEWS



First Baseline Distribution

STACK VIEWS



Last Baseline Distribution

STACK VIEW MAGIC

- When you hide one view, the others will redistribute themselves accordingly
- You can even animate this change